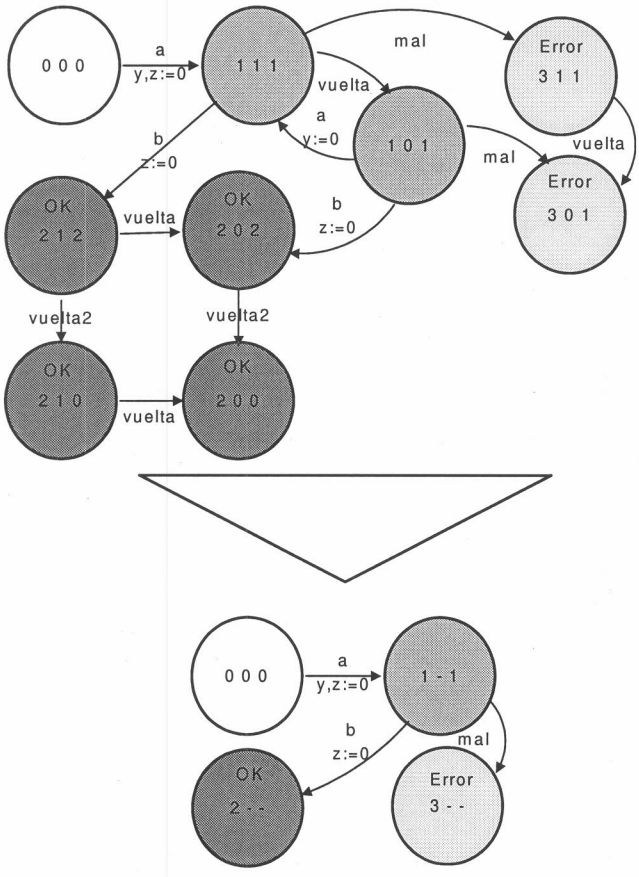
TESIS DE LICENCIATURA

Un método de reducción para la composición de sistemas temporizados



Diego Garbervetsky

Directores: Dr. Alfredo Olivero, Lic. Victor Braberman

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

2000

2000

Resumen

Los sistemas temporizados están presentes en una gran variedad de campos tales como control industrial, fabricación, robótica, aeronavegación, trenes, automóviles, comunicaciones, multimedia, medicina, etc. Estos sistemas embebidos controlan dispositivos que realizan tareas de las cuales pueden depender vidas humanas y/o importantes inversiones económicas. Generalmente son sistemas críticos donde es vital satisfacer requerimientos temporales duros (donde tarde significa mal). La necesidad de predecir el comportamiento temporal de este tipo de sistemas ha llevado a un gran desarrollo de teorías y herramientas formales para la especificación, diseño y verificación de sistemas donde el tiempo lleva un papel preponderante. Entre estos métodos formales podemos encontrar el uso de autómatas temporizados [AD94].

Los autómatas temporizados, son un formalismo conocido y con alta aceptación para la modelización de este tipo de sistemas. Una característica importante de los mismos es que permiten trabajar de forma modular modelando al sistema completo mediante la composición de sus módulos (también llamado producto). Se han desarrollado lógicas que permiten expresar propiedades y herramientas que posibilitan su verificación automática las cuales se han utilizado con éxito en casos reales.

Lamentablemente el proceso de verificación es muy costoso. Se ha probado que la verificación es un problema $PSPACE$ [ACD93] que depende del tamaño del autómata, la cantidad de relojes que utiliza y el valor de las constantes que aparecen en las condiciones de sus transiciones. Como consecuencia, en muchos casos el problema de la verificación es intratable.

Para atacar este problema definimos un método de reducción del tamaño para sistemas temporizados ideado para ser aplicado en el momento de la composición de los mismos. En la práctica, se utiliza frecuentemente la técnica de anexar a un sistema, formado por varios componentes, un componente virtual llamado *observador* sobre el cual luego se realizan consultas a fin de verificar ciertas propiedades del sistema en general. Aprovechando esta característica, nuestro método reduce el sistema de manera tal que nuestro producto reducido obtenido no se distinga del producto estándar al menos para la verificación de propiedades predicadas sobre el *observador*. Para ello se analiza para cada locación de este autómata que componentes participan realmente o tienen incumbencia en el futuro.

Hemos implementado un prototipo basado en estas ideas y hemos visto que en la mayoría de los casos que pueden encuadrarse en la técnica descripta,

el método obtiene una reducción muy significativa en el tamaño del sistema y como consecuencia en el tiempo a la hora de verificar las propiedades.

Una característica adicional de este esquema es que es ortogonal a otros esquemas de reducción por lo cual puede combinarse con estos para obtener una reducción aún mayor.

Agradecimientos

En primer lugar quiero agradecer a la Universidad todo lo que me ha dado, ya que fue por varios años mi segunda casa. Espero poder devolverle todo esto con creces mediante mi aporte como docente. En especial quiero a agradecer a Mónica Bobrowski por ser la persona que me inicio y me acompaño en mis pasos como docente en la facultad.

A mis directores Victor y Alfredo por todo el apoyo que me han dado y por su infinita voluntad para ayudarme cada vez que lo necesite durante el desarrollo de este trabajo.

Al jurado por haber aceptado el compromiso de revisar y corregir este trabajo.

A todos mis amigos de facultad. A Lau y Mariela, eternas compañeras de estudio y de la vida. A Pabli, y Ari Camporeale dos personas increiblemente buenas y dispuestas. A Ari Dembling por estar siempre cuando lo necesite. A Sergín y Techas mis amigos y compañeros de parranda. A Marce, que pude conocerla gracias a la facu.

A mis amigos de toda la vida, Juanjo, Groc, Fede y Diego por todos los momentos vividos y los que viviremos.

A Caro por todo el amor y paciencia que me tuvo en esta última etapa.

A mis compañeros de trabajo, que me bancaron y me dieron libertad para poder finalizar este trabajo libremente.

A mi familia, por todo el cariño, amor y apoyo que me dieron en todo momento.

En general, quiero extender mi agradecimiento a todas las personas que directa o indirectamente siempre me apoyaron y me dieron cariño.

Todo vuelve...

Índice General

Capítulo 1

Introducción

Los sistemas temporizados están presentes en una gran variedad de campos tales como control industrial, fabricación, robótica, aeronavegación, trenes, automóviles, comunicaciones, multimedia, medicina, etc. Estos sistemas embebidos controlan dispositivos que realizan tareas de las cuales pueden depender vidas humanas y/o importantes inversiones. Generalmente son sistemas críticos donde es vital satisfacer requerimientos temporales duros (donde tarde significa mal). La necesidad de predecir el comportamiento temporal de este tipo de sistemas ha llevado a un gran desarrollo de teorías y herramientas formales para la especificación, diseño y verificación de sistemas donde el tiempo lleva un papel preponderante.

Los autómatas temporizados [AD94], son un formalismo conocido y con alta aceptación para la modelización de este tipo de sistemas. Básicamente estos son máquinas de estados finitos equipadas con un conjunto de relojes. Se han desarrollado herramientas tales como KRONOS, UPPALL, HYTECH [DOTY96, BLL+96, HHWT95] y lógicas que posibilitan la verificación automática propiedades (ver por ejemplo [ACD93]) y fue utilizado con éxito en casos reales (ver por ejemplo [CW96]). Una característica importante de este formalismo es que permiten especificar de forma modular generando el sistema completo mediante la composición de sus módulos.

El proceso de verificación de propiedades sobre estos sistemas se lo conoce como Model Checking y ha sido profundamente estudiado (por ejemplo [BCMD90] para el sistemas no temporizados y [HXSY94] para sistemas temporizados). Lamentablemente este proceso es muy costoso en el caso de los sistemas temporizados. Se ha probado que la verificación es un problema $PSPACE$ [ACD93] que depende del tamaño del autómata, la cantidad de relojes y el valor de las constantes que se utilizan en las condiciones del autómata. Como consecuencia, en muchos casos el problema de la verificación es intratable.

1.1 Trabajos realizados anteriormente

Para enfrentar al alto costo de la verificación de sistemas temporizados se han desarrollado varios esquemas de reducción que apuntan a disminuir el tamaño o simplificar al sistema a verificar.

Como ejemplo podemos citar el trabajo *"Reducing the number of clock variables of timed automata"* presentado en [DY96]. Dicho trabajo propone un método para reducir el número de relojes de un autómata temporizado combinando dos algoritmos. El primero de ellos consiste en la detección de los relojes activos, o sea, los relojes relevantes para la evolución del sistema. El segundo detecta conjuntos de relojes que son siempre iguales. Se utilizan bisimulaciones temporales fuertes como herramienta teórica de abstracción para definir cuando dos autómatas tienen el mismo comportamiento. En el trabajo se muestran resultados empíricos que demuestran que la utilización de los resultados provistos por el método pueden ser utilizados para reducir tanto el tiempo y el espacio utilizado en el proceso de verificación de propiedades.

Otro ejemplo que podemos citar es *"Analysis based on time abstracting bisimulation"* presentado en [TY96] que presenta una herramienta para la reducción de un autómata temporizado abstrayendo la parte temporal del mismo. Para formalizar el método de reducción, se definen las nociones de bismulación abstrayendo el tiempo fuerte y débil. La primera es una generalización temporal del grafo de región presentado en [AD94] y preserva la estructura branch del sistema. La segunda es menos restrictiva pero puede no preservar la estructura branch. En el trabajo se define un algoritmo para calcular el modelo finito más grueso, a partir del sistema de transiciones infinito, que es minimal módulo la bisimulación que abstrae el tiempo.

En [TKY+98] se presenta un método basado en autómatas temporizados para calcular los delays en circuitos combinatorios. Se deriva, a partir de la estructura topológica del circuito, una partición de la red y la correspondiente descomposición conjuntiva del OBDD que representa el espacio de estados. El algoritmo que computa el delay opera sobre esta representación y obtiene, sobre cierta clase de circuitos, una performance que es órdenes de magnitud mejor que un algoritmo no especializado de recorrida.

En [DT98] podemos encontrar un estudio del uso de abstracciones como un medio para reducir el tiempo de verificación (en el momento de verificación) de consultas de alcanzabilidad en sistemas de tiempo real y en [KTAB96] se propone una herramienta para verificar abstracciones, lo que permite proponer abstracciones y constatar su correctitud respecto al sistema original.

1.2 Nuestro enfoque

Nuestro trabajo apunta a la reducción del tamaño del sistema temporizado. Para ello definimos un método de reducción ideado para ser aplicado en el momento de la composición de los mismos. En la práctica se utiliza frecuentemente la técnica de anexar a un sistema, formado por varios componentes, un componente virtual llamado *observador* (ver por ejemplo [BF99, ECJ97, KLK96]) sobre el cual luego se realizan consultas de alcanzabilidad a fin de verificar ciertas propiedades del sistema en general. Aprovechando esta característica nuestro método reduce el sistema de manera tal que nuestro producto reducido no se distinga del producto estándar al menos para la verificación de propiedades predicadas sobre el *observador*. Para ello construimos una *función de relevancia* que indica para cada locación del *observador* que componentes participan realmente o tienen incumbencia en el futuro. En el momento de la composición utilizamos esta función para determinar, para cada locación del *observador*, que componentes influyen en el resultado de la verificación y cuales ya no lo hacen, eliminando de la composición a estos últimos a partir de esa locación.

Hemos implementado un prototipo basado en estas ideas y hemos visto que en la mayoría de los casos que pueden encuadrarse en la técnica descripta y construyendo adecuadamente el *observador*, el método obtiene una reducción muy significativa con la consecuente reducción en los tiempos de verificación.

Nuestro método es novedoso por el hecho de que explota una característica no aprovechada hasta el momento, la relevancia o actividad de las componentes, y además de conseguir una reducción significativa, es ortogonal a los otros métodos conocidos, por lo que podríamos aplicar este método combinado con los otros.

1.3 Organización de la Tesis

En el capítulo 2 realizaremos las definiciones preliminares necesarias para el desarrollo de nuestra teoría. En el capítulo 3 extenderemos esta teoría definiendo las nociones de *componentes Input/Output* y *bisimulación observacional continua*, demostrando que la misma preserva la lógica $TCTL$. En el capítulo 4 definiremos la *función de relevancia* indicando cuales son sus requisitos. Luego definimos como es la *composición paralela utilizando relevancia* y probamos que el sistema compuesto utilizando nuestro método es observacional continuo bisimilar al compuesto de manera estándar y, por lo tanto, respeta fórmulas $TCTL$ que predican sobre el *observador*. También presentamos un algoritmo para realizar la composición utilizando relevan-

cia junto con un algoritmo para calcular una aproximación de la función de relevancia. Finalizamos el capítulo presentando algunas pequeñas mejoras posibles al método. En el capítulo 5 analizamos algunos ejemplos para validar el impacto práctico de nuestro método. Por último en el capítulo 6 presentamos nuestras conclusiones y los posibles trabajos que podrían encararse en el futuro.

Capítulo 2

Definiciones preliminares

En este capítulo presentamos las nociones fundamentales de la teoría de autómatas temporizados. Para un estudio más detallado sobre esta teoría, sus lógicas asociadas y su chequeo, ver por ejemplo [Yov98, HXSY94, ACD93].

2.1 Autómata Temporizado

Un autómata temporizado es una máquina de estados finitos equipada con un conjunto de relojes. Los relojes son variables reales positivas que son usadas para describir el tiempo transcurrido entre eventos. Todos los relojes están sincronizados, es decir todos avanzan al mismo tiempo. Sin embargo los relojes pueden tener diferentes valores ya que estos pueden ser reseteados individualmente al atravesar por una transición. Las transiciones tienen asociadas guardas que son predicados sobre el espacio de los relojes. Las guardas determinan cuando una transición puede ser tomada.

2.1.1 Definiciones preliminares

- $X = \{x_1, x_2, \ldots, x_n\}$ es un conjunto de relojes
- Una *valuación* es una función $v : X \mapsto \Re_{\geq 0}$ donde $v(x_i)$ es el valor asociado al reloj x_i.
- Dado X un conjunto de relojes, $\rho \subseteq X$ y una valuación v definimos $Reset_\rho(v)$ como:

$$Reset_\rho(v)(x) = \begin{cases} 0 & \text{si } x \in \rho, \\ v(x) & \text{en caso contrario.} \end{cases}$$

  Llamamos V_X al conjunto $[X \stackrel{tot}{\to} \Re_{\geq 0}]$ de funciones totales de X a $\Re_{\geq 0}$

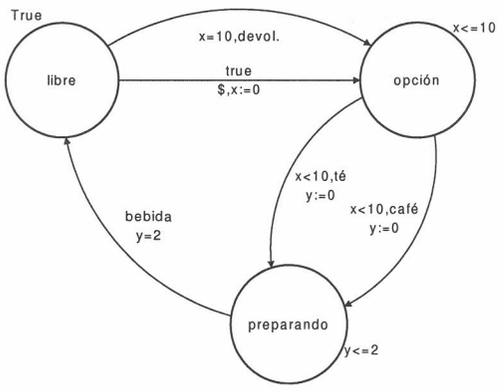


Figura 2.1: Un autómata temporizado que modela una expendedora de bebidas.

- Dada $v \in V_X$ y $\delta \in \Re_{\geq 0}$ llamamos $v + \delta$ a la valuación que asigna a cada reloj $x \in X$ el valor $v(x) + \delta$.
- Dado X un conjunto de relojes definimos el conjunto de restricciones sobre relojes Ψ_X, según la siguiente gramática:

$$\psi ::= x \prec c | x - x' \prec c | \psi \wedge \psi | \neg\psi$$

donde $x, x' \in X, \prec \in <, \leq$ y $c \in \mathbb{N}$

Las restricciones son evaluadas contra las valuaciones de los relojes. Una valuación v es satisfecha por una restricción $\psi \in \Psi_X$ (lo escribimos $v \models \psi$) si

$$\begin{array}{lll} v \models x \leq c & \iff & v(x) \leq c \\ v \models x - x' \leq c & \iff & v(x) - v(x') \leq c \\ v \models \psi \wedge \psi' & \iff & v \models \psi \wedge v \models \psi' \\ v \models \neg\psi & \iff & v \not\models \psi \end{array}$$

Llamamos $[\psi]$ al conjunto de valuaciones que satisfacen ψ, o sea,

$$[\psi] = \{v \in V_X | v \models \psi\}$$

2.1.2 Autómata temporizado

Definición 2.1.1 (Automata Temporizado) *Un autómata temporizado es una tupla* $A =< S, X, \Sigma, E, I, s_0 >$ *donde*

1. *S es un conjunto finito de locaciones.*
2. *X es un conjunto finito de relojes.*
3. *Σ es un conjunto de etiquetas.*
4. *E es un conjunto finito de aristas. Cada arista $e \in E$ es una tupla $< s, \sigma, \psi, \alpha, s' >$ donde:*
   (a) *$s \in S$ es la locación de origen*
   (b) *$s' \in S$ es la locación de destino*
   (c) *$\sigma \in \Sigma$ es una etiqueta*
   (d) *$\psi_x \in \Psi_X$ es una restricción de habilitación de la arista*
   (e) *$\alpha \in 2^X$ indica que relojes son reseteados al atravesar la arista.*
5. *$I : S \stackrel{tot}{\rightarrow} \Psi_X$ es una función total que asocia a cada locación una restricción sobre los relojes. Lo referimos como $I(s)$ y es llamado el Invariante de la locación.*
6. *$s_0 \in S$ es la locación inicial.*

Notación 1 *Dado $A =< S, X, \Sigma, E, I, s_0 >$ definimos:*

$Loc(A) = S$

$Clocks(A) = X$

$Trans(A) = E$

$Labels(A) = \Sigma$

$Inv(A) = I$

$Init(A) = s_0$

Dado $t =< s, e, \psi, \alpha, s' > \in E$ definimos :

$source(t) = s$

$Label(t) = e$

$Guard(t) = \psi$

$Reset(t) = \alpha$

$dest(t) = s'$

Diremos que $s \xrightarrow{e}_A s'$ *si* $\exists t \in Trans(A)$ *tal que* $source(t) = s$, $dest(t) = s'$ *y* $label(t) = e$.

Dada $t \in Trans(A)$ *diremos que es un* stutter *o* loop back *si* $source(t) = s$, $dest(t) = s$ *y* $Reset(t) = \emptyset$.

2.1.3 Semántica

Semántica informal

Un estado para un autómata temporizado A es un par (s, v) donde $s \in S$ y $v \in V_X$ y $v \models I(s)$.

La arista $a =< s, e, \psi_a, \alpha, s' >$ puede ser atravesada desde el estado (s, v) si $v \models \psi_a$. El estado resultante es (s', v') donde $v' = Reset_\alpha(v)$.

Los valores de los relojes crecen *uniformemente* con el tiempo.

El tiempo avanza solo en las locaciones.

Pasar por una arista no avanza los relojes.

El sistema puede permanecer en la locación s siempre y cuando $v \models I(s)$.

Semántica formal

Dado un autómata $A =< S, X, \Sigma, E, I, s_0 >$ definimos $G =< Q, Q^0, \rightarrow, \Sigma >$ un sistema de transiciones etiquetado (LTS) donde

- $Q = \{(s, v) | s \in S \wedge v \in V_X \wedge v \models I(s)\}$ es un conjunto de estados.
- $\rightarrow \subseteq Q \times (\Sigma \cup \Re_{\geq 0}) \times Q$ una relación de transición con dos tipos de transiciones:
  1. *Temporales*: el paso de un tiempo t es representado por una transición etiquetada con t.

$$\frac{v + t' \models I(s), 0 \leq t' \leq t}{(s, v) \xrightarrow{t} (s, v + t)}$$

  2. *Instantáneas*: dadas por la ejecución de una acción e. La transición lleva la etiqueta e.

$$\frac{< s, e, \psi, \alpha, s' > \in E, v \models \psi, Reset_\alpha(v) \models I(s')}{(s, v) \xrightarrow{e} (s', Reset_\alpha(v))}$$

Notación 2 *Dado* $q = (s, v)$ *escribimos:*

- $q + t = (s, v')$ *donde* $\forall x \in X \Rightarrow v'(x) = v(x) + t$.
- $q^{@} = s$
- $q(i) = v(x_i)$

Llamaremos también stutter *a la transición temporal* $q \xrightarrow{0} q$.

Secuencias de ejecución y Non-zenoness

Una ejecución o corrida σ para un autómata temporizado A es una secuencia infinita de estados y transiciones.

$$\sigma = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} \dots$$

donde $(\forall i \in \mathbf{N} \Rightarrow q_i \in Q \wedge l_i \in (\Sigma \cup \Re_{\geq 0}))$. Llamaremos $R_A(q)$ al conjunto de ejecuciones que comienzan en q y $R_A = \bigcup_{q \in Q} R_A(q)$ al conjunto de todas las corridas de A.

Dada $\sigma \in R_A$ una posición es un par $(i, t) \in I\!N \times \Re_{\geq 0}$ tal que:

- $t \leq l_i$ si $l_i \in \Re_{\geq 0}$ o
- $t = 0$ si $l_i \in \Sigma$.

Llamaremos $\Pi(\sigma)$ al conjunto de todas las posiciones de la secuencia σ. Definimos un orden total $<<$ para $\Pi(\sigma)$ de la siguiente forma:

$$(i, t) \ << \ (j, t') \iff i < j \vee (\ i = j \wedge t \leq t'\ )$$

Dada una posición $(i, t) \in \Pi(\sigma)$ su estado correspondiente es:

$$\sigma(i, t) = q_i + t$$

Para $\sigma \in R_A$ y $i \in \mathbf{N}$ definimos $\tau_\sigma(i) = \tau_\sigma(i, 0)$ como el tiempo transcurrido entre el estado q_0 y el estado q_i. Se define intuitivamente $\tau(i, t)_\sigma$ de la siguiente manera:

$$\tau_\sigma(0, t) = t$$

$$\tau_\sigma(i+1,t) = \tau_\sigma(i,t) + \begin{cases} 0 & \text{si } l_i \in \Sigma, \\ l_i & \text{en caso contrario.} \end{cases}$$

Una ejecución $\sigma \in R_A$ es divergente en el tiempo si $lim_{i\to\infty}\tau_\sigma(i,0) = \infty$. Llamaremos $R_A^\infty(q)$ al conjunto de las ejecuciones divergente que empiezan en q.

Diremos que un autómata A es *Non-Zeno* si todo prefijo finito de sus corridas es prefijo de una secuencia en R_A^∞.

Dada $\sigma = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} \ldots$

Un estado q' es alcanzable desde el estado q si este pertenece a alguna ejecución que empieza por q. Definimos $Reach_A(q)$ al conjunto de estados alcanzable desde q. Esto es:

$$Reach_A(q) = \{q' \in Q | \exists \sigma \ \in R_A(q) \wedge \exists i \in \mathbf{N} \wedge \sigma(i,0) = q'\}$$

2.1.4 Composición del Autómata Temporizado

El formalismo de autómatas temporizados permite modelar componentes individualmente para luego modelar un sistema como la composición de los componentes individuales.

Definición 2.1.2 (Composición paralela) *Dados los autómatas* $A_1 =< S_1, X_1, \Sigma_1, E_1.I_1, s_{0_1} >$ *y* $A_2 =< S_2, X_2, \Sigma_2, E_2, I_2, s_{0_2} >$ *donde* $X_1 \cap X_2 = \emptyset$

Sea $\epsilon = \Sigma_1 \cap \Sigma_2$. *La composición paralela* $A_1||A_2$ *se define como el autómata* $A =< S_1 \times S_2, X_1 \cup X_2, \Sigma_1 \cup \Sigma_2, E, I, < s_{0_1}, s_{0_2} >>$ *tal que:*

- $<< s_1, s_2 >, e, \psi, \alpha, < s_1', s_2' > > \in E \iff$
  - $< s_1, e, \psi, \alpha, s_1' > \in E_1 \wedge e \notin \epsilon \wedge s_2 = s_2'$ *ó*
  - $< s_2, e, \psi, \alpha, s_2' > \in E_2 \wedge e \notin \epsilon \wedge s_1 = s_1'$ *ó*
  - $< s_i, e, \psi_i, \alpha_i, s_i' > \in E_i \wedge e \in \epsilon \wedge \psi = (\psi_1 \wedge \psi_2) \wedge \alpha = \alpha_1 \cup \alpha_2$
- $I(< s_1, s_2 >) = I(s_1) \wedge I(s_2)$

Nota: La composición es asociativa y conmutativa, es decir $A_0||A_1 = A_1||A_0$ y $A_0||A_1||A_2 = A_0||(A_1||A_2)$ [1].

[1] La igualdad es en realidad abuso de notación ya que en realidad son isomorfos

Notación 3 *Dado* $A = A_0||A_1||\ldots||A_n$, *una transición* $< s, e, \psi, \alpha, s' > \in Trans(A)$ *escribimos:*

$\pi_i(s) = s_i$, $\pi_i(s') = s'_i$ *(la componente iésima de la locación)*

$\pi_i(\psi) = \psi_i$ *(la condición correspondiente a la arista de la iésima componente. Si la componente no participa, es True)*

$\pi_i(\alpha) = \alpha_i$ *(el reset correspondiente a la arista de la iésima componente. Si la componente no participa, es* $\emptyset$*)*

Semántica de la composición paralela

A continuación caracterizaremos las transiciones del LTS asociado al autómata compuesto. Debido a su construcción, el comportamiento del mismo es totalmente dependiente del funcionamiento de cada componente individual.

Notación 4 *Sea* $q =< s, \vec{v} >$ *un estado de un autómata compuesto, definimos:*

$\pi_i^{@}(< s, \vec{v} >) = \pi_i(s)$

$\pi_i^{R}(< s, \vec{v} >) = \{v(x)/x \in Clocks(A_i)\}$

$\pi_i(q) =< \pi_i^{@}(q), \pi_i^{R}(q) >$

Sea $A = A_0||A_1||\ldots||A_n$ y G_i los LTS asociados a A_i.

- *Transiciones Temporales*:

$$\frac{(\forall i)0 \leq i \leq n : \pi_i(q) \xrightarrow{t}_i \pi_i(q+t)}{q \xrightarrow{t} q+t}$$

- *Transiciones Instantáneas*:

$$\frac{(\forall i)0 \leq i \leq n : a \notin \Sigma_i \wedge \pi_i(q) = \pi_i(q') \vee (a \in \Sigma_i \wedge \pi_i(q) \xrightarrow{a}_i \pi_i(q'))}{q \xrightarrow{a} q'}$$

2.2 La lógica TCTL

TCTL (Timed Computational Tree Logic), presentada en [ACD93], es una extensión temporal de la bien conocida lógica CTL introducida por [CE81].

Sea $\mathcal{I}$ un conjunto de intervalos en $\Re^+$ de la forma $[c, c']$, $[c, c')$, $(c, c']$, (c, c'), (c, ∞) y $[c, \infty)$, donde c, c' son números naturales. Una fórmula TCTL se define siguiendo las siguientes reglas:

$$\phi \ ::= p \ \mid \ \neg\phi \ \mid \ \phi \vee \phi \ \mid \ \exists\phi\mu_I\phi \ \mid \ \forall\phi\mu_I\phi$$

donde $p \in Props$ es una proposición atómica, $s \in Loc(A)$ y $I \in \mathcal{I}$ es un intervalo.

Sea A un autómata temporizado con el conjunto de locaciones S. Sea $P : Props \mapsto 2^S$ una función que asocia a cada proposición atómica un conjunto de locaciones de A. Dada una fórmula ϕ en $TCTL$ y un estado (s, v), la relación *satisface* $(s, v) \models_P \phi$ se define intuitivamente sobre la sintaxis de ϕ de la siguiente forma:

$$\begin{array}{lcl}
(s,v) \models_P p & \iff & s \in P(p) \\
(s,v) \models_P \neg\phi & \iff & \neg((s,v) \models \phi) \\
(s,v) \models_P \phi_1 \vee \phi_2 & \iff & (s,v) \models \phi_1 \ \vee \ (s,v) \models \phi_2 \\
(s,v) \models_P \exists\phi_1\mu_I\phi_2 & \iff & \exists\sigma \in R_A^\infty(s,v).\exists k \in \Pi(\sigma).\ \sigma(k) \models (\phi_2 \wedge \tau_\sigma(k)) \\
 & & \wedge\forall k' << k.\ \sigma(k') \models \phi_1 \vee (\phi_2 \wedge \tau_\sigma(k') \in I) \\
(s,v) \models_P \forall\phi_1\mu_I\phi_2 & \iff & \forall\sigma \in R_A^\infty(s,v).\exists k \in \Pi(\sigma).\ \sigma(k) \models (\phi_2 \wedge \tau_\sigma(k)) \\
 & & \wedge\forall k' << k.\ \sigma(k') \models \phi_1 \vee (\phi_2 \wedge \tau_\sigma(k') \in I)
\end{array}$$

Dada una fórmula $TCTL$ ϕ el conjunto de estados que la satisface es llamado conjunto característico de ϕ y es escrito como $[\phi]$. Diremos que un autómata A satisface ϕ, y lo escribimos $A \models_P \phi$, si todos los estados de A satisfacen ϕ.

2.3 Bisimulaciones

Las bisimulaciones son relaciones de equivalencia que son utilizadas para identificar sistemas con comportamientos similares. Entre otros usos, son utilizadas como base teórica para demostrar analogías en el comportamiento de sistemas más abstractos con respecto al sistema original. Las equivalencias en comportamiento basadas en relaciones de bisimulación han sido utilizadas con éxito para la verificación de sistemas concurrentes. Básicamente diremos que un sistema es bisimilar a otro, con respecto a una relación B, si el primero puede simular al segundo y viceversa.

A continuación presentaremos las nociones de bisimulación temporal fuerte y débil estudiadas por ejemplo en [Č92, NRSV90, LW97, ECJ97, Yi90],etc. Las mismas son una extensión para sistemas temporizados de las nociones presentadas en [Mil89].

Bisimulación temporal fuerte

La noción de bisimulación fuerte fue estudiada por ejemplo en [NRSV90]. Básicamente define que un sistema es fuertemente temporal bisimilar a otro si ambos tienen el mismo comportamiento tanto lo respectivo a las trazas (o sea a la parte no temporal) como en lo respectivo al tiempo y ha sido demostrado que tiene la característica de preservar $TCTL$.

Definición 2.3.1 (Bisimulación temporal fuerte) *Sean* $G1 =< Q_1, Q_1^0, \rightarrow_1, \Sigma >$ *y* $G2 =< Q_2, Q_2^0, \rightarrow_2, \Sigma >$ *los LTS de dos TA. Una relación* $B \subseteq Q_1 \times Q_2$ *es una* bisimulación *sí y solo sí para todo par* $(q_1, q_2) \in B$, $a \in (\Re_{\geq 0} \cup \Sigma)$ *se cumple:*

1. *Si* $q_1 \xrightarrow{a}_1 q_1'$, *entonces* $q_2 \xrightarrow{a}_2 q_2'$ *y* $(q_1', q_2') \in B$.
2. *Si* $q_2 \xrightarrow{a}_2 q_2'$, *entonces* $q_1 \xrightarrow{a}_1 q_1'$ *y* $(q_1', q_2') \in B$.

Dos estados q_1 y q_2 son bisimilares, y lo escribimos $q_1 \simeq q_2$, si existe una bisimulación B y $(q_1, q_2) \in B$.

Dos autómatas temporizados son bisimilares si sus estados iniciales son bisimilares.

Bisimulación temporal débil

La noción de bisimulación temporal débil fue estudiada por ejemplo en [LW97, Yi90, Č92], etc. Este tipo de bisimulación tiene la característica de abstraer las acciones, es decir, se concentra sólo en los aspectos temporales del sistema y no en aspectos atemporales (transiciones instantáneas).

Definición 2.3.2 *Dado un* $t \in \Re_{\geq 0}$, *y dos estados* q_0 *y* q_n , $q_0 \longrightarrow^t q_n$ *si y sólo si existe una secuencia finita* $\sigma = q_0 \xrightarrow{l_0} q_1 \xrightarrow{l_1} \ldots q_n$ *con* $l_i \in \Sigma \cup \Re_{\geq 0}$ *tal que* $\tau_\sigma(n) = t$

Definición 2.3.3 (Bisimulación temporal débil) *Dados los LTS* $G_1 =< Q_1, Q_1^0, \rightarrow_1, \Sigma >$ *y* $G_2 =< Q_2, Q_2^0, \rightarrow_2, \Sigma >$ *asociados a dos TAs, una relación entre estados* $B \subseteq Q_1 \times Q_2$ *es una bisimulación temporal débil si se cumple que para todo* $a, a' \in Labels(A)$, $t \in \Re_{\geq 0}$,

1. *Si* $p \xrightarrow{a}_1 p'$ *entonces para algún* q', q_1', $q \longrightarrow^0 q_1'$ *y* $(p', q_1') \in B$ *o* $q \longrightarrow^0 q_1' \xrightarrow{a'}_2 q'$, *y* $(p', q') \in B$.

2. *Si* $p \xrightarrow{t}_1 p'$ *entonces , para algún* q', $q \longrightarrow^t q'$ *y* $(p', q') \in B$.

3. *Se cumplen (1) y (2) si se invierten los roles de* p *y* q.

Capítulo 3

Extensiones a la teoría de los sistemas temporizados

3.1 Interfases Input/Output entre autómatas

Nótese que el modelo clásico de composición de autómatas temporizados es simétrico, ya que cuando dos o más autómatas sincronizan por medio de alguna etiqueta, no se indica qué rol cumple esa etiqueta en cada autómata. Para aplicar el método que proponemos necesitamos un modelo que ofrezca mayor información sobre la relación entre los componentes del sistema. Para ello necesitamos declarar una interfase para los autómatas que divida las etiquetas en Inputs, Outputs e internas. Consideramos a este modelo bastante general e intuitivo y en general a la mayoría de los sistemas modelados con TAs se les puede declarar una interfase I/O.

Cuando se modela el comportamiento de un sistema mediante la composición de sus componentes individuales, la comunicación entre estas es mediante las aristas que sincronizan. En general, cuando esto sucede una componente estará emitiendo un *Output* que será "escuchado"por una o varias componentes que estarán realizando un *Input*.

Podemos encontrar como trabajos relacionados al uso de I/O por ejemplo el de [KTAB96] donde los autores presentan la noción de Procesos Temporales no bloqueantes o el de [GSSAL94] que introduce la noción de "*Live Timed Automata*". En [BS00] se definen condiciones temporales para que las sincronizaciones sean no bloqueantes. Nuestra versión es una simplificación de la que fue presentada en [Bra00].

Definición 3.1.1 (Interfases admisibles de Input/Output) *Dado un TA non-zeno* A, $I, O \subseteq Labels(A)$ *es una interfase admisible input/output para*

A si y sólo si:

1. $I \cap O = \emptyset$ *(debe existir una separación entre las etiquetas de entrada y salida).*
2. *Para todo estado alcanzable q y para todo $l \in I$, existe un q' tal que $q \xrightarrow{l} q'$ (toda etiqueta de Input esta siempre habilitada).*
3. *Para todo estado alcanzable q existe una corrida divergente r que empieza en q y no contiene ningún label de Input (los Inputs son no bloqueantes. Siempre puede avanzar el tiempo sin necesidad de tomar una transición de Input. Esto es más fuerte que non-zeno ya que esta última no depende de las transiciones de Input).*
4. *Para cada $l \in Labels(A)$ tal que $l \in O$ se cumple que cada corrida que contiene un numero infinito de ocurrencias de l es divergente en el tiempo (el TA es "fuertemente non-zeno" [TY99] para Outputs).*

Los conjuntos I, O son llamados Input y Output de A. Diremos que si una etiqueta pertenece a los Inputs o a los Outputs de A entonces es exportada por A.

Definición 3.1.2 (Componente) *Un componente es una tupla $< A, (I, O) >$ donde A es un autómata temporizado y (I, O) es una interfaz admisible de Input/Output para A.*

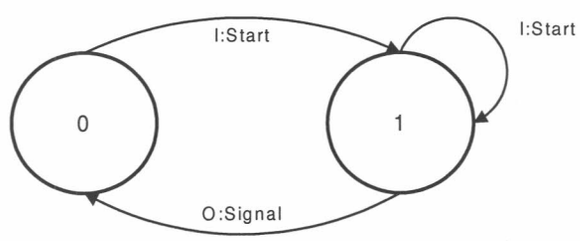


Figura 3.1: Una componente I/O.

Definición 3.1.3 (Componentes compatibles) *Dados dos componentes $C1 =< A, (I, O) >$ y $C2 =< A', (I', O') >$, estos son compatibles si y solo si*

1. *Todas las labels de $Labels(A) \cap Labels(A')$ son exportadas en $C1$ y en $C2$.*

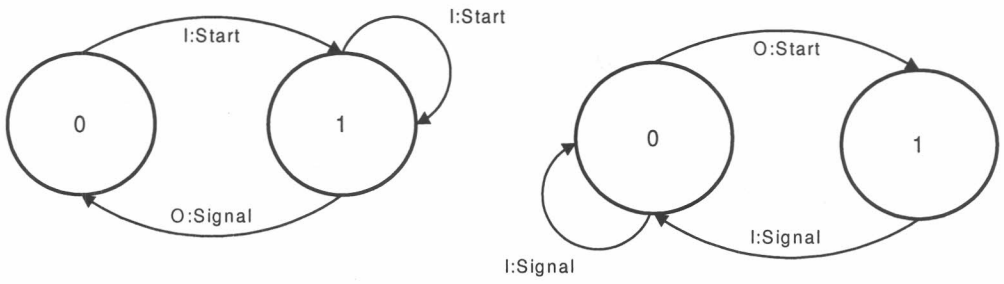


Figura 3.2: Dos componentes I/O compatibles.

2. $O \cap O' = \emptyset$

Lema 3.1.1 *Sean* $< A, (I, O) >$ *y* $< A', (I', O') >$ *dos componentes compatibles. Entonces* $< A||A', ((I \cup I)' - (O \cup O'), (O \cup O') >$ *es también un componente.*

Demostración: La parte más complicada de la demostración es que $A||A'$ sea non-zeno sin tomar en cuenta las transiciones de Input. Sea q un estado alcanzable [1] para una corrida en $A||A'$ y sean $q_A = \pi_0(q)$ y $q_{A'} = \pi_1(q)$. Sea $k \in \Re+$ una constante. Por la definición de componente, sabemos que deben existir corridas r, r' comenzando respectivamente en q_A y $q_{A'}$ de tiempo mayor que k tal que r no contiene ninguna I-transición y r' no contiene ninguna I'-transición (por lo que la corrida en $A||A'$ no contendría ninguna etiqueta en Input de $A||A'$).

A continuación mostraremos un procedimiento para obtener una corrida para $A||A'$ a partir de r y r'. Para ello aparearemos r y r'. Si las transiciones instantáneas de r y r' son ordenadas de acuerdo al tiempo de ocurrencia, es fácil combinarlas y construir una corrida para $A||A'$ ($r_{A||A'} = q_0 \rightarrow \ldots \rightarrow q_i$) hasta encontrar la primer transición o, etiquetada como Output. Esta transición debe ser compartida por el otro componente. Sin pérdida de generalidad, supongamos que la primera transición compartida de Output aún no sincronizada proviene de r por lo cual $o \in O$. Describamos esta transición como $\pi_0(q_i) \xrightarrow{o}_A s$. Sea $i \in I'$ una etiqueta que coincide con o. por definición de Input, existe una transición que tiene a i' como etiqueta y está siempre habilitada en A' por lo que debe existir una transición $\pi_1(q_i) \xrightarrow{i'}_{A'} s'$. Por lo tanto podemos sincronizar está transición, colocarla en la corrida de

[1] Al hablar de alcanzable nos referimos a cualquier corrida, no necesariamente divergente

$A||A'$ como $q_{i+1} = (s, s')$ y repetir todo este procedimiento a partir de (s, s'). Notar que debemos realizar este ajuste sólo cuando nos encontramos con transiciones de Output. Pero por la definición de componente si existe una corrida con infinitas transiciones de Output, el tiempo debe diverger, por lo que podemos asegurar que podremos construir una corrida de tiempo mayor que k con una cantidad finita de ajustes.

Bajo este procedimiento convertimos las transiciones Output e Input que sincronicen en transiciones de Output de la nueva componente (esto lo hacemos para que está componente pueda ser compuesta nuevamente). Debido a esto el conjunto de Input de la nueva componente será $((I \cup I)' - (O \cup O')$ y el conjunto de Output será $(O \cup O')$. □

3.2 Bisimulación observacional continua

En el capítulo anterior presentamos las nociones de bisimulación temporal fuerte y débil (ver definiciones 2.3.1 y 2.3.3). Como mencionamos antes, la primera de estas tiene la ventaja de preservar $TCTL$, sin embargo suele ser muy restrictiva, ya que deben respetarse tanto transiciones temporales como instantáneas.

Las bisimulaciones temporales débiles son menos restrictivas, ya que para que dos estados sean bisimilares sólo deben respetar los tiempos y no es necesario que lo hagan con las transiciones instantáneas. Lamentablemente este tipo de bisimulación, aunque se respeten las asignaciones proposicionales (esto es, $(p, q) \in B \Rightarrow [p]_{\mathcal{P}} = [q]_{\mathcal{P}}$), no necesariamente preserva $TCTL$ ya que no se preserva la estructura branch del sistema. Este problema también acontece en el caso no temporal y tampoco se preserva la lógica CTL. En la figura 3.3 podemos ver dos LTS que son temporal débil bisimilares, respetando las asignaciones proposicionales y las etiquetas. Sin embargo, la fórmula CTL $\Big((\exists \diamond C)\ \exists\ \mu\ B\Big)$ que consulta si existe una secuencia que llegue a B en el cual para todas las posiciones anteriores se cumple B o se cumple que hay una secuencia (a partir de esa posición) que llegue a C, es verdadera para el primer sistema y no lo es para el segundo.

La razón del problema puede ser explicada en el par bisimilar débil $(3, 4')$. Mientras que en el primer sistema 3 cumple con la propiedad ya que desde 1 se llega a B y C, en el segundo para llegar a $4'$ y cumplir con B es necesario pasar por $2'$ donde ya no puede cumplirse C. Este problema se debe a que la bisimulación débil no exige bisimilaridad en los estados intermedios.

A continuación presentaremos una nueva noción de bisimulación que soportará a nuestro método de reducción. La bisimulación que definiremos es más débil que la bisimulación temporal fuerte definida previamente, pero es

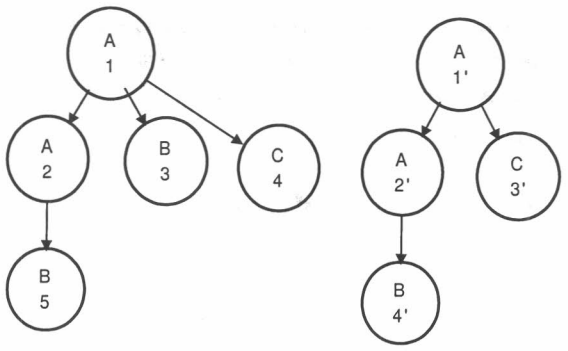


Figura 3.3: Dos autómatas débil bisimilares con $B = \{(1,1'),(2,2'),(3,4'),(4,3'),(5,4')\}$

más fuerte que la bisimulación temporal débil, ya que preserva la estructura branch del sistema temporal.

El propósito de esta bisimulación será que dos autómatas observacional continuo bisimilares satisfacerán el mismo conjunto de fórmulas $TCTL$.

La definición presentada es una simplificación adaptada para este método de las nociones presentadas en [Bra00] y es similar a la noción de *branching time bisimulation* encontradas en [GW89, dNMV90] para el caso no temporizado y [Yov93] para el caso temporizado, la principal diferencia con respecto a esta última noción es que la nuestra es muy poco más restrictiva ya que exigimos bisimilaridad en todos los estados intermedios.

Definiremos cuando un autómata puede simular a otro con respecto a su estructura branch respetando los tiempos transcurridos.

Definición 3.2.1 *Sean* $G_1 =< Q_1, Q_1^0, \rightarrow_1, \Sigma >$ *y* $G_2 =< Q_2, Q_2^0, \rightarrow_2, \Sigma >$ *los LTS correspondiente a dos TAs.*

Dado un $t \in \Re_{\geq 0}$, *una relación entre estados* $B \subseteq Q_1 \times Q_2$ *y dos estados* $p \in Q_1$ *y* $q_0 \in Q_2$ *tal que* $(p, q_0) \in B$ *el estado* q_0 *tiene una transición observacionalmente-*λ *hacia* q_n *de tamaño* t *con respecto a* B *y* p *(que escribimos* $q_0 \stackrel{B,p}{\longrightarrow}^t q_n$*) sí y solo sí existe una secuencia finita* $\sigma = q_0 \stackrel{l_0}{\rightarrow}_2 q_1 \stackrel{l_1}{\rightarrow}_2 \ldots q_n$ *con* $l_i \in \Sigma \cup \Re_{\geq 0}$ *tal que* $\tau_\sigma(n) = t$ *y para cada posición* k *en* $\Pi(\sigma)$ *tal que* $p \stackrel{\tau_\sigma(k)}{\rightarrow}_1 p + \tau_\sigma(k)$, *entonces* $(p + \tau_\sigma(k), \sigma(k)) \in B$.

Definición 3.2.2 (Bisimulación Observacional Continua) *Dados los LTS* $G_1 =< Q_1, Q_1^0, \rightarrow_1, \Sigma >$ *y* $G_2 =< Q_2, Q_2^0, \rightarrow_2, \Sigma >$ *asociados a sus respectivos TAs, dos asignaciones de variables proposicionales* $P_i : Props \mapsto 2^{Q_i}$,

$i = 1..2$ [2]*, entonces una relación entre estados $B \subseteq Q_1 \times Q_2$ es una bisimulación observacional continua con respecto a P_1 y P_2 si $(p,q) \in B$ implica que $[p]_{P_1} = [q]_{P_2}$ y se cumple que para todo $a, a' \in \Sigma)$, $t \in \Re_{\geq 0}$,*

1. *Si $p \xrightarrow{a}_1 p'$ entonces para algún q', q_1, $q \xrightarrow{B,p}{}^{0} q_1$ y $(p', q_1) \in B$ o $q \xrightarrow{B,p}{}^{0} q_1 \xrightarrow{a'}_2 q'$, y $(p', q') \in B$.*

2. *Si $p \xrightarrow{t}_1 p'$ entonces , para algún q', $q \xrightarrow{B,p}{}^{t} q'$ y $(p', q') \in B$.*

3. *Se cumplen (1) y (2) si se invierten los roles de p y q.*

Dos autómatas temporizados A_1, A_2 son observacional continuo bisimilares con respecto a P_1y P_2 (denotado $A_1 \simeq^{P_1,P_2} A_2$) si y solo si existe una bisimulación observacional continua tal que sus estados iniciales sean bisimilares.

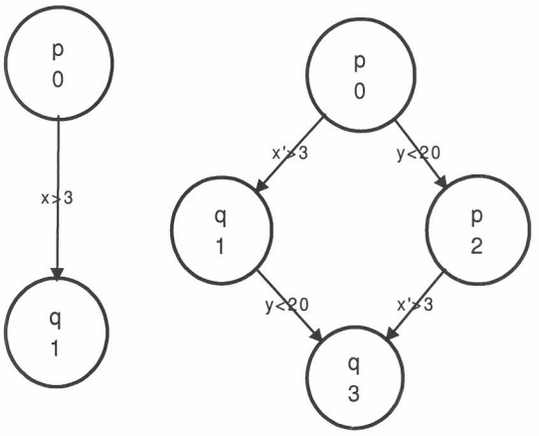


Figura 3.4: Dos autómatas observacional continuo bisimilares con $B = \{(0,0),(0,2),(1,1),(1,3)\}$

La idea detrás de la bisimulación observacional continua es que, a diferencia de la bisimulación temporal débil, los estados inscriptos dentro de la secuencia utilizada para simular a la transición del sistema comparado, deben mantener cierta "correspondencia" con los estados del sistema simulado. Esta restricción permite que ambos sistemas conserven una importante propiedad, que será demostrada en la siguiente sección.

[2] $Q_i^{@}$ representa las locaciones correspondientes a los estados de G_i

3.2.1 Conservación de $TCTL$

Tal como lo indicamos previamente, nuestro propósito es mostrar que nuestra bisimulación preserva $TCTL$. Nuestra demostración está basada en la que se utilizó en [CGL94] para fórmulas CTL.

A continuación definiremos la noción de correspondencia que es necesaria para esta demostración. La idea detrás de esta definición es que dos corridas se corresponden con respecto a una bisimulación si cada posición de una corrida puede ser *correspondida* por otra posición de la otra corrida manteniendo el orden entre las posiciones.

Definición 3.2.3 (Correspondencia) *Dados dos LTS, G_1 y G_2 y una bisimulación B, diremos que una corrida r' de G_2 está en correspondencia con una corrida r en G_1 si y sólo si existe un mapeo total y suryectivo C de las posiciones de r' a las posiciones de r $(C : \Pi(r') \mapsto \Pi(r))$ tal que:*

- *Para cada $k' \in \Pi(r')$ se cumple $(r(C(k')), r'(k')) \in B$ y $\tau_r(C(k')) = \tau_r(k')$ (se respeta el tiempo y la bisimulación).*
- *Si $k'_1 <<_{r'} k'_2$ entonces $C(k'_1) <<_r C(k'_2)$.*

Lema 3.2.1 *Sean p y q dos estados observacional continuo bisimilares. Entonces para cada corrida r que empieza por p existe una corrida r' en correspondencia que empieza por q.*

Demostración: Lo probaremos de forma constructiva. Sea B una BOC sobre los estados de G_1 y G_2 y sea $r = p \xrightarrow{l_0} p_1 \xrightarrow{l_1} \ldots$ una corrida en G_1.

Debido a la definición de la bisimulación observacional continua:

- Si l_0 es una etiqueta, entonces r' puede tener una secuencia de transiciones comenzando por q tal que cada posición es bisimilar a p y finalmente saltar a q_1 (o no hacerlo si $(p_1, q) \in B$, tomando $q_1 = q$) siendo este un estado tal que $(p_1, q_1) \in B$ (esto es, $q \xrightarrow{B,p}{}^{0} w \xrightarrow{l'_0}_2 q_1$ o directamente $q \xrightarrow{B,p}{}^{0} q_1$). Entonces podemos mapear todas las posiciones de r' desde q hasta la posición de q_1 (sin incluirla) a la posición $(0,0)$ de r y mapear la posición de q_1 en r' con la posición $(1,0)$ de r .
- Si $l_0 \in \Re_{\geq 0}$ podemos aplicar un procedimiento similar ya que podemos hallar un q_1 tal que $q \xrightarrow{B,p}{}^{t} q_1$ y $(p_1, q_1) \in B$. En este caso podemos mapear cada posición k de r' desde q hasta la posición de q_1 (sin incluirla) con la posición $(0, \tau'_r(k))$ de r y mapear la posición de q_1 en r' con la posición $(1,0)$ de r.

Luego construimos un r' y su correspondencia con r para una transición. Podemos aplicar el mismo razonamiento para $p1$ y $q1$. El punto fijo de este procedimiento es la secuencia r' en correspondencia con r. □

Lema 3.2.2 *Dados los LTS G_1, G_2 respectivos a los TAs A_1, A_2, dos asignaciones de variables proporcionarles $P_i : Props \mapsto 2^{Loc(A_i)}$, $i = 1..2$, y una fórmula TCTL ϕ en Props. Sea $\simeq^{P_1,P_2}$ una bisimulación observacional continua con respecto a P_1 y P_2 y dos estados p, q tal que $p \simeq^{P_1,P_2} q$. Entonces $p \models_{P_1} \phi \iff q \models_{P_2} \phi$.*

Demostración: Probaremos el lema utilizando inducción en la estructura de ϕ.

Base:

$$p \models_{P_1} pr \iff p^{@} \in P_1(pr) \iff_{(\simeq \text{ respeta la asig. de props.})} q^{@} \in P_2(pr) \iff q \models_{P_2}$$

Paso inductivo:

Existen varios casos. Tomaremos uno interesante, en los otros casos el razonamiento es similar.

$\Rightarrow$)

$p \models_{P_1} \exists \phi_1 \mu_I \phi_2 \iff \exists r \in R^{\infty}_{A_1}(p)$ y $\exists k \in \Pi(r)$ tal que $\tau_r(k) \in I$ y $r(k) \models_{P_1} \phi_2 \wedge \forall m << k.\ r(m) \models_{P_1} \phi_1 \vee (\phi_2 \wedge \tau_r(m) \in I)$.

Sabemos por el lema 3.2.1 que existe $r' = q \xrightarrow{l_0} q_1 \xrightarrow{l_1} \ldots \in R^{\infty}_{A_2}(q)$ en correspondencia con r. En particular, por suryectividad de la correspondencia, existe una posición k' de r' tal que $C(k') = k$. Entonces $r(k) \simeq^{P_1,P_2} r'(k')$ y $\tau_r(k) = \tau_{r'}(k') \in I$ y, por hipótesis inductiva, $r'(k') \models_{P_2} \phi_2$. Más aún, para todo $m' << k'$ se cumple que $r(C(m')) \simeq^{P_1,P_2} r'(m')$ y como $C(m') << C(k') = k$, entonces, por hipótesis inductiva, $r'(m') \models_{P_2} \phi_1 \vee (\phi_2 \wedge \tau_{r'}(m') \in I)$.

Luego, $q \models_{P_2} \exists \phi_1 \mu_I \phi_2$.

$\Leftarrow$)

$q \models_{P_2} \exists \phi_1 \mu_I \phi_2 \iff \exists r \in R^{\infty}_{A_2}(q)$ y $\exists k \in \Pi(r)$ tal que $\tau_r(k) \in I$ y $r(k) \models_{P_2} \phi_2 \wedge \forall m << k.\ r(m) \models_{P_2} \phi_1 \vee \phi_2$.

La demostración es simétrica a la anterior ya que q es bisimilar a p (podemos encontrar una secuencia r en correspondencia con r').

□

Teorema 3.2.1 *Dados dos TA $A_i (i = 1, 2)$, dos asignaciones de variables*

proporcionarles $P_i : Props \mapsto 2^{Loc(A_i)}$, $i = 1, 2$, *y una fórmula* $TCTL$ ϕ *en* $Props$.

Si $A_1 \simeq^{P_1,P_2} A_2$ *entonces* $A_1 \models_{P_1} \phi \iff A_2 \models_{P_2} \phi$.

Demostración: Como $A_1 \simeq^{P_1,P_2} A_2$, sus estados iniciales son CO-Bisimilares. Luego por el lema 3.2.2 sabemos que estos estados satisfacen las mismas fórmulas $TCTL$. □

Capítulo 4

Relevancia de Autómatas

En este capítulo desarrollaremos formalmente nuestro método. Para ello primero definiremos formalmente que es una función de relevancia. Luego presentaremos una redefinición de la composición paralela que aplicará la función de relevancia y probamos que la misma es observacional continuo bisimilar a la composición estándar. Una vez demostrada esta importante propiedad, presentaremos un algoritmo para realizar la composición paralela y otro para calcular una función de relevancia que cumpla con los requisitos especificados en su definición formal. Por último presentamos algunos refinamientos en la teoría y en los algoritmos que mejoran la eficacia del método.

4.1 Definición Formal

En esta sección definiremos que es una función de relevancia, indicando las condiciones mínimas que debe cumplir. La noción de relevancia esta basada en la noción de influencia entre autómatas que presentaremos a continuación.

Definición 4.1.1 (Influencia) *Dada una componente I/O A y un TA o componente B*

$A \xrightarrow{inf}_S B$ *con* $S \subseteq Loc(B)$ *si* :

$Ouput(A) \cap \{Label(t) : t \in Trans(B)/source(t) \in S \wedge (source(t) \neq dest(t) \vee reset(t) \neq \emptyset)\} \neq \emptyset$

Que una componente influencie a un autómata/componente cuando el segundo este en alguna locación de S, significa que si el primero toma alguna

acción, el segundo, estando dentro del entorno de S, se verá afectado y por lo tanto modificará su estado (ver figura 4.1).

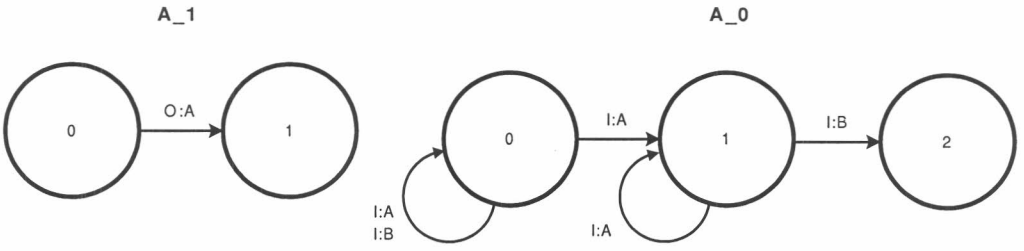


Figura 4.1: $A\_1 \stackrel{inf}{\rightarrow}_S A\_0$ si y solo si $0 \in S$.

Definición 4.1.2 *Dado un conjunto de componentes $A_1, \ldots, A_n$ y el TA o componente A_0 y $q \in Loc(A_0)$.*

$S_q^*(A_i) = \{a_i \in Loc(A_i)/(q, a_1, \ldots, a_n)$ *es alcanzable en el LTS asociado a* $A_0||A_1|| \ldots ||A_n\}$

Podemos sobreestimar este conjunto de varias maneras. Por ejemplo $\{a_i \in Loc(A_i)/(q, a_1, \ldots, a_n) \in Loc(A_0||A_1|| \ldots ||A_n)\}$, o $\{a_i \in Loc(A_i)/(q, a_i) \in Loc(A_0||A_i)\}$

El conjunto $S_q^*(A_i)$ refleja las posibles locaciones en donde puede encontrarse el componente A_i cuando el autómata A_0 se encuentra el la locación q.

Observación 4.1 $S_q^*(A_0) = \{q\}$.

Ahora estamos en condiciones de formalizar la noción de relevancia.

Definición 4.1.3 (Relevancia) *Sea $\{0 \ldots n\}$ un conjunto de indices. Dados las componentes I/O $A_1, \ldots, A_n$ y un componente* observador *A_0.*

Definimos a $Rel : Loc(A_0) \mapsto 2^{\{0 \ldots n\}}$ como una función que indica que autómatas/componentes son relevantes para una locación determinada de A_0

Sean $q, q' \in Loc(A_0)$, $i, k \in \{0 \ldots n\}$. $i \in Rel(q)$ si y solo si se cumple alguna de las siguientes condiciones:

1. *$i = 0$.*

2. $q \xrightarrow{b}_{A_0} q' \wedge i \in Rel(q')$ *(es cerrada hacia atrás).*

3. $k \in Rel(q) \wedge A_i \overset{inf}{\rightarrow}_{S_q^*(A_k)} A_k$ *(es transitiva).*

Observación 4.2 *El ítem 1 de la definición de relevancia asegura que A_0 sea siempre relevante.*

Observación 4.3 *El ítem 2 implica que $Rel(q_{i+1}) \subseteq Rel(q_i)$ donde q_{i+1} representa una locación sucesora a q_i viendo al autómata como un grafo dirigido. Esta propiedad es muy importante ya que nos permite olvidar a los autómatas una vez que ya no son relevantes (ver figura 4.2).*

Observación 4.4 *El ítem 3 indica que una componente es relevante si influencia a un autómata relevante en las locaciones en donde este puede estar para la locación q del autómata A_0. Que lo influencia significa que el estado de este se ve afectado por lo que haga el otro autómata. Es por ello que no tomamos en cuenta las aristas que son stutter (ver figura 4.2).*

4.2 Composición aplicando el criterio de relevancia

A continuación definiremos formalmente la composición teniendo en cuenta la relevancia de las componentes según el observador A_0.

Para ello definiremos primero una función que dado un estado del autómata compuesto nos devuelve un estado donde solo aparecen los estados locales de los autómatas relevantes.

$$\begin{array}{lll} P_{Rel} & : (n+1) - \text{tupla de S} \rightarrow & (n+1) - \text{tupla de (S} \cup \{\perp\})s \\ P_{Rel} & (< q, a_1, \ldots, a_n >) = & < q, b_1, \ldots, b_n > / \\ & & b_i = \begin{cases} a_i \;\; si \;\; i \in Rel(q) \\ \perp \; en \; caso \; contrario \end{cases} \end{array}$$

Ahora podemos definir fácilmente la composición aplicando *Rel* de la siguiente manera:

$$A_0 || \equiv_{Rel} A_1 || \ldots || A_n = A_r = (S_r, X_r, \Sigma_r, E_r, I_r, s_{0_r})$$

siendo:

$$S_r = \{P_{Rel}(s) |\ s \in Loc(A_0) \times Loc(A_1) \times \ldots \times Loc(A_n)\}$$

$$X_r = \bigcup_{i=0}^{n} Clocks(A_i)$$

$$I_r(s_r) = \bigwedge_{\{i:0..n/i \in Rel(\Pi_0(s))\}} Inv(A_i)(s_r)$$

$$\Sigma_r = \bigcup_{i=0}^{n} Labels(A_i)$$

$$\begin{aligned} E_r = \{ \quad & < P_{Rel}(s), e_r, \psi_r, \alpha_r, P_{Rel}(s') > \quad | < s, e_r, \psi, \alpha, s' > \in Trans(A_0 || A_1 || \ldots || A_n) \\ & \wedge\ e_r \in \bigcup_{\{i:0..n/i \in Rel(\Pi_0(s))\}} \{e \in Labels(A_i)\} \\ (*) \qquad & \wedge\ (\nexists i : 0..n)(i \notin Rel(\Pi_0(s)) \wedge e_r \in Output(A_i)) \\ & \wedge\ \psi_r = \bigwedge_{\{i:0..n/i \in Rel(\Pi_0(s))\}} \pi_i(\psi) \\ & \wedge\ \alpha_r = \bigcup_{\{i:0..n/i \in Rel(\Pi_0(s))\}} \pi_i(\alpha)\ \} \end{aligned}$$

(*) Notar que colocamos esta condición para eliminar los Inputs que podrían quedar colgantes si alguna/s de las componentes que actuaban como Output deja/n de ser relevante/s. De todos modos, este ajuste solo elimina transiciones superfluas, que en realidad no afectan el comportamiento del autómata compuesto, ya que por la observación [4.4] sabemos que estas aristas deben ser "stutters".

En la figura 4.2 se muestran el autómata compuesto de la manera estándar y el autómata reducido. Los diferentes tipos de sombreado de las locaciones representan la equivalencia entre ellas según la noción de relevancia (P_{Rel}). Como se puede observar, cada conjunto de locaciones equivalentes es traducido en una única locación en el autómata reducido. En la tabla 4.1 podemos observar la función de relevancia y los conjuntos $S_q^*(A_k)$ para este ejemplo. Para comprender fácilmente el significado de los conjuntos $S_q^*(A_k)$ hay que observar el autómata compuesto de forma estándar de la siguiente forma: elegir el valor de q (la locación en el observador), luego elegir A_k (el autómata a analizar) y tomar el valor de la k-ésima posición de las locaciones en las cuales la 0-iésima locación coincide con q.

Analicemos este sistema: Por definición A_0 es siempre relevante, ya que es el observador. A_1 es relevante en la locación 0 ya que $A_0 \in Rel(0)$ y $A_1 \xrightarrow{inf}_{S_0^*(A_0)} A_0$ (A_0 utiliza en la locación 0 una transición con etiqueta a que es un Output en A_1). A_2 es relevante en la locación 1 ya que $A_0 \in Rel(1)$

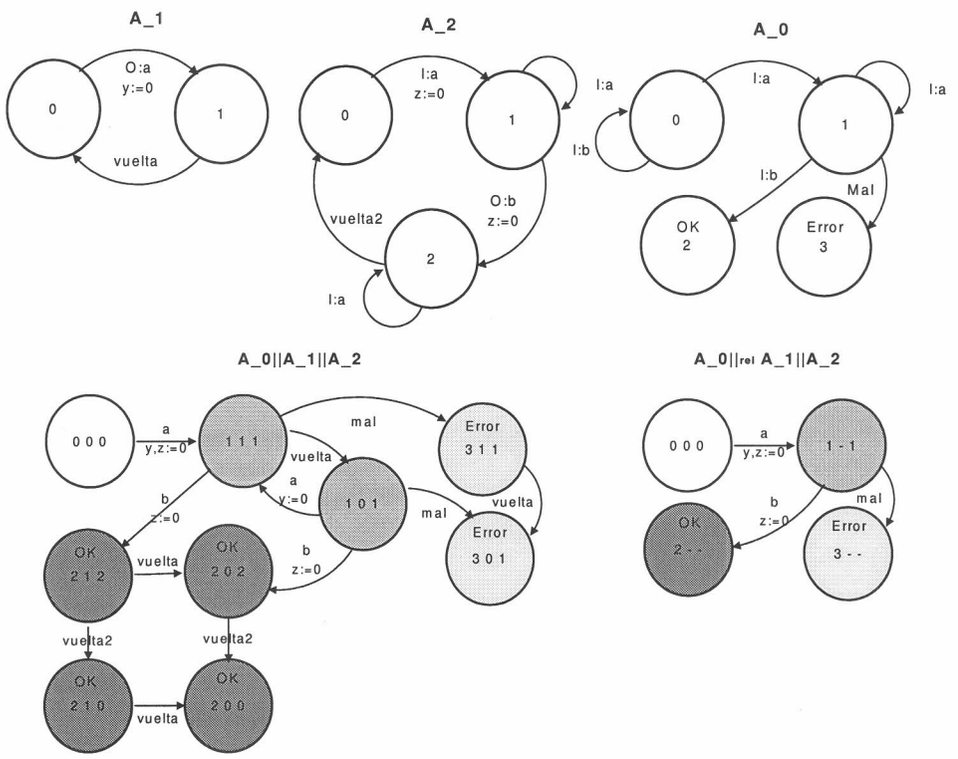


Figura 4.2: Composición Estándar vs. Reducida. Las locaciones sombreadas representan las clases de equivalencia.

y $A_2 \overset{inf}{\rightarrow}_{S_1^*(A_0)} A_0$ (A_0 utiliza en la locación 1 una transición con etiqueta b que es un Output en A_2). Luego por el item 2 de la definición de relevancia, necesariamente A_2 es relevante en la locación 0. En cambio A_1 no es relevante en la locación 1 ya que $\neg(A_1 \overset{inf}{\rightarrow}_{S_1^*(A_0)} A_0)$ y $\neg(A_1 \overset{inf}{\rightarrow}_{S_1^*(A_2)} A_2)$. Bajo el mismo razonamiento obtenemos que en las locaciones 2 y 3 solo A_0 es relevante.

Locación	**Relevantes**	$S_q^*(A_0)$	$S_q^*(A_1)$	$S_q^*(A_2)$
0	A_0, A_1, A_2	$S_0^*(A_0) = 0$	$S_0^*(A_1) = 0$	$S_0^*(A_2) = 0$
1	A_0, A_2	$S_1^*(A_0) = 1$	$S_1^*(A_1) = 0, 1$	$S_1^*(A_2) = 1$
2	A_0	$S_2^*(A_0) = 2$	$S_2^*(A_1) = 0, 1$	$S_2^*(A_2) = 0, 2$
3	A_0	$S_3^*(A_0) = 3$	$S_3^*(A_1) = 0, 1$	$S_3^*(A_2) = 1$

Tabla 4.1: Cálculo de relevancia para el ejemplo de la figura 4.2

En la siguiente sección veremos que el autómata compuesto reducido es observacional continuo bisimilar al autómata compuesto de la manera

estándar.

Semántica del autómata compuesto por relevancia

A continuación definiremos la semántica del autómata compuesto. De la misma manera que en el caso de la composición estándar, el comportamiento del mismo depende de las componentes individuales, pero en este caso el autómata compuesto es menos restrictivo ya que algunas componentes dejan de tener importancia a medida que avanzamos por las locaciones del componente *observador* (A_0).

Sea $A = A_0||_{\equiv_{Rel}} A_1||\ldots||A_n$. Sean G_i los LTS asociados a A_i. Caracterizamos las transiciones del LTS de A de la siguiente forma:

- *Temporales*:

$$\frac{(\forall i) 0 \leq i \leq n : i \in Rel(\pi_0^{@}(s)) \Rightarrow \pi_i(s) \xrightarrow{t}_i \pi_i(s'))}{s \xrightarrow{t} s'}$$

- *Instantáneas*:

$$a \in \bigcup_{0 \leq i \leq n : i \in Rel(\pi_0^{@}(s))} \Sigma_i \wedge (\nexists i : 0..n)(i \notin Rel(\Pi_0(s)) \wedge e_r \in Output(A_i)) \wedge$$

$$\frac{(\forall i)\Big(0 \leq i \leq n : (a \notin \Sigma_i \vee i \notin Rel(\pi_0^{@}(s)) \wedge \pi_i(s) = \pi_i(s')) \vee (a \in \Sigma_i \wedge \pi_i(s) \xrightarrow{a}_i q \wedge (\pi_i(s') = q) \vee \pi_i(s') = \bot)\Big)}{s \xrightarrow{a} s'}$$

4.3 Equivalencia entre Autómata reducido y el compuesto

A continuación demostraremos que nuestro método de composición respeta $TCTL$ con respecto al observador. Es decir, dada una fórmula $TCTL$ que solo consulta alguna propiedad sobre el observador, se obtendrá el mismo resultado tanto en la composición estándar como en la composición reducida. Este será el resultado más importante de este trabajo ya que es el que garantiza el funcionamiento de nuestro método de reducción.

Teorema 4.3.1 *Dados $A_1, \ldots, A_n$ componentes I/O, A_0 un componente observador, $P : Props \mapsto 2^{Loc(A_0)}$ una asignación de variables proposicionales para A_0. Si Rel es una función de relevancia entonces:*

$$A_0||A_1||\ldots||A_n \simeq^{P^*,P'^*} A_0||_{\equiv_{Rel}} A_1||\ldots||A_n$$

donde P^* y P'^* son las extensiones naturales de P sobre $A_0||A_1||\ldots||A_n$ y $A_0||_{\equiv_{Rel}} A_1||\ldots||A_n$ respectivamente.[1]

Demostración: Sean $A = A_0||A_1||\ldots||A_n$ y $A' = A_0||_{\equiv_{Rel}} A_1||\ldots||A_n$. Para demostrar el teorema trabajemos con los LTS asociados a A y A'. Sean $G_i =< K_i, K_i^0, \rightarrow_i, \Sigma_i >$ los LTS asociados a los A_i y sean $G_A =< K_A, K_A^0, \rightarrow_A, \Sigma >$ y $G_{A'} =< K_{A'}, K_{A'}^0, \rightarrow_{A'}, \Sigma >$ los LTS asociados con A y A' que se comportan de acuerdo a las definiciones [2.1.4] y [4.2] respectivamente.

A continuación propondremos nuestra relación B de la siguiente manera:

Dados $s1 \in K_{G_A}, s1' \in K_{G_{A'}}$

$$(s1, s1') \in B \iff (\forall i)\Big(0 \leq i \leq n : \pi_i^{@}(s1') \neq \perp \Rightarrow \pi_i(s1) = \pi_i(s1')\Big)$$

Debemos probar que B es una bisimulación observacional continua. Para ello primero demostraremos que, utilizando B, A puede simular A' y luego probaremos que A' puede simular A usando la misma relación B.

$\Rightarrow$)

Esta parte de la demostración es la más sencilla debido a que A es más restrictivo que A', por lo que es fácil ver que A' puede simular a A.

Sean $s1, s2 \in K_A, s1' \in K_{A'}$, tal que $(s1, s1') \in B$.

- Si $s1 \xrightarrow{t}_A s2$

  Por la semántica de la composición para A definida en [2.1.4] sabemos que $(\forall i)(0 \leq i \leq n : \pi_i(s1) \xrightarrow{t}_i \pi_i(s2))$. Entonces, por la semántica de A' definida en [4.2] y porque $(s1, s1') \in B$, también podemos hacer $s1' \xrightarrow{t}_{A'} s2'$ con $(s2, s2') \in B$.

  Notar que en este caso se da trivialmente que $s1' \xrightarrow{B,s1^t} s2'$.

[1] P^* es una extensión natural de P sobre $A_0||A_1||\ldots||A_n$ si para cada $p \in P^*$, se cumple que $(q, a_1, \ldots, a_n) \in P^*(p)$ si y sólo si $q \in P(p)$. Es razonamiento es análogo para P'^*

- Si $s1 \xrightarrow{e}_A s2$

  Por la semántica de la composición para A definida en [2.1.4] sabemos que $(\forall i)(0 \leq i \leq n : e \notin \Sigma_i \wedge \pi_i(s1) = \pi_i(s2) \vee (e \in \Sigma_i \wedge \pi_i(s1) \xrightarrow{e}_i \pi_i(s2))$. Sabiendo que $(s1, s1') \in B$, veamos que ocurre con A' según la semántica definida en [4.2].

  - Si $(e \in \bigcup_{0 \leq i \leq n : i \in Rel(\pi_0^{@}(s1))} \Sigma_i)$ podemos hacer $s1' \xrightarrow{e}_{A'} s2'$ (ya que coinciden en las componentes relevantes) y concluir, por la observación [4.3] en la def. del Rel, que $(s2, s2') \in B$

    Nota: Podría suceder que e pertenezca a $Input(A_i)$ con A_i relevante pero que la componente A_k que exporta e como Output no sea relevante. En este caso, está transición no existirá en A' ya que en la definición de la composición se eliminan los Inputs colgantes. De todos modos por la observación [4.4] A_i debía hacer Stutter en A, por lo que podemos simular está transición haciendo $s1' \xrightarrow{0}_{A'} s2'$.

  - Si $(e \notin \bigcup_{0 \leq i \leq n : i \in Rel(\pi_0^{@}(s1))} \Sigma_i)$ estamos ante una transición en la que solo actúan componentes no relevantes. Por lo tanto basta con simularla haciendo $s1' \xrightarrow{0}_{A'} s2'$ y concluir, por la observación [4.3] en la def. del Rel, que $(s2, s2') \in B$

  Notar que estos casos se ajustan respectivamente a $s1' \xrightarrow{B, s1^0} q1' \xrightarrow{e}_2 s2'$ y $s1' \xrightarrow{B, s1^0} s2'$

$\Leftarrow$)

Esta parte de la demostración es un poco más compleja debido a que A' es menos restrictivo que A, por lo que no es tan simple ver que A puede simular a A'.

Sean $s1', s2' \in K_{A'}, s1 \in K_A$, tal que $(s1, s1') \in B$.

- Si $s1' \xrightarrow{t}_{A'} s2'$

  Por la semántica de la composición definida en [4.2] sabemos que $(\forall i)(0 \leq i \leq n : i \in Rel(\pi_0^{@}(s1')) \Rightarrow \pi_i(s1') \xrightarrow{t}_i \pi_i(s2'))$. El problema es que no podemos asegurar que el invariante de las componentes no relevantes les permita avanzar t unidades sin realizar ningún salto discreto. Podría suceder que, antes de t unidades, las componentes irrelevantes nos obliguen a salir de $s1^{@}$.

  En el caso general podría suceder lo siguiente:

$r = q1 \xrightarrow{a_{1_1}} \ldots \xrightarrow{a_{1_j}} q1' \xrightarrow{t1} q1'+t1 \xrightarrow{a_{2_1}} q2 \ldots \xrightarrow{a_{2_j}} q2' \xrightarrow{t2} q2'+t2 \xrightarrow{} q3 \ldots \xrightarrow{} qn-1'+tm \; qm = s2$

con $s1 = q1$, $\sum_{i=0}^{n} t_i = t$ y a_{i_j} una transición interna o de output (se podrían evitar las de Input ya que estas no son obligatorias, porque por definición siempre existe una corrida divergente sin transiciones de Input).

La corrida tiene la propiedad que $(\forall m)(0 \leq k \leq m : (\forall i)(0 \leq i \leq n | i \in Rel(\pi_0^{@}(q_k)) \Rightarrow \pi_i(q_k) = \pi_i(q'_k)))$ ya que

- si $\xrightarrow{a_{i_j}}$ es una transición interna (como sabemos irrelevante) no se modifica el estado de las componentes relevantes y además por ser interna no influencia a otros autómatas.
- si $\xrightarrow{a_{i_j}}$ es una transición (irrelevante) de output (por la observación [4.4] en la def. de Rel) necesariamente si esta sincroniza con alguna componente relevante, la transición de esta última debe ser un stutter (sino en $G_{A'}$ abandonaría la locación $s1'^{@}$ y por lo tanto $\xrightarrow{a_{i_j}}$ pertenecería a una componente relevante, siendo esto absurdo).

Podemos reescribir la secuencia como $s1 \xrightarrow{B, s1'^{t}} s2$ ya que para toda posición k de la secuencia se cumple que $(r(k), s1' + \tau_r(k)) \in B$.

Luego sabemos que $(qm, s1' + t) \in B$ y ya que $qm = s2$ y $s2' = s1' + t$, podemos concluir que $(s2, s2') \in B$.

- Si $s1' \xrightarrow{e}_{A'} s2'$

Por la semántica de la composición para A definida en [4.2] sabemos que:

$e \in \bigcup_{0 \leq i \leq n : i \in Rel(\pi_0^{@}(s1'))} \Sigma_i \;\wedge\; (\nexists i : 0..n)(i \notin Rel(\Pi_0(s1')) \wedge e_r \in Output(A_i)) \;\wedge (\forall i)\Big(0 \leq i \leq n : (e \notin \Sigma_i \vee i \notin Rel(\pi_0^{@}(s1')) \wedge \pi_i(s1') = \pi_i(s2')) \vee (a \in \Sigma_i \wedge \pi_i(s1') \xrightarrow{e}_i q \wedge (\pi_i(s2') = q) \vee \pi_i^{@}(s2') = \bot)\Big)$

En el caso de las componentes relevantes, no hay nada que decir ya que también estarán habilitadas en A y su comportamiento es idéntico al de la semántica de la composición normal. Debemos observar que ocurre con las componentes no relevantes que exportan e.

- Si A_k es no relevante necesariamente $e \notin Output(A_k)$ ya que, por definición, si $e \in Output(A_k)$, esta componente debe ser relevante.
- Si $e \in Input(A_k)$, por definición la transición de esta componente estará siempre habilitada.

Luego podemos hacer $s1 \xrightarrow{e}_A s2$ y por la observación [4.4] de la def. de *Rel* concluir que $(s2, s2') \in B$.

Notar que este caso se ajusta a $s1 \xrightarrow{B,s1'^0} q1' \xrightarrow{e}_1 s2$.

Ahora debemos analizar que ocurre con las variables proposicionales. Sabemos que A_0 participa en A y en A' y que A_0 es siempre relevante (por la observación [4.2] en la def. de *Rel*). Luego podemos concluir que si $(s1, s1') \in B$ entonces $[s1]_{P^*} = [s1']_{P'^*}$.

Luego B es una bisimulación observacional continua.

Ahora nos falta solo probar que los estados iniciales de A y A' son observacional continuo bisimilares.

Sean $G_A =< K_A, K_A^0, \rightarrow_A >$ y $G_{A'} =< K_{A'}, K_{A'}^0, \rightarrow_{A'} >$ los LTS asociados a A y A', sean $s_A^0 =< Init(A_0), \ldots, Init(A_n) >, s_{A'}^0 = P_{Rel}(< Init(A_0), \ldots, Init(A_n) >)$. Luego los estados iniciales de G_A y $G_{A'}$ son:

$K_A^0 = \{< s_A^0, \vec{0} >\}$ y $K_{A'}^0 = < s_{A'}^0, \vec{0} >\}$.

Claramente $\forall s1 \in K_A^0 \, , s1' \in K_{A'}^0 \Rightarrow (s1, s1') \in B$.

Luego $A_0||A_1|| \ldots ||A_n \simeq^{P^*,P'^*} A_0||_{\equiv_{Rel}} A_1|| \ldots ||A_n$ como queríamos demostrar. □

Corolario 4.3.1 *Dados el componente observador A_0 y los componentes $A_1, \ldots, A_n$ y dado $P : Props \mapsto 2^{Locs(A_0)}$. Sean P^* y P'^* las extensiones naturales de P sobre $A_0||A_1|| \ldots ||A_n$ y $A_0||_{\equiv_{Rel}} A_1|| \ldots ||A_n$ respectivamente.*

Para toda fórmula $TCTL$ ϕ que se cumple:

$$A_0||A_1|| \ldots ||A_n \models_{P^*} \phi \iff A_0||_{\equiv_{Rel}} A_1|| \ldots ||A_n \models_{P'^*} \phi$$

Demostración: Sea $A = A_0||A_1|| \ldots ||A_n$ y $A' = A_0||_{\equiv_{Rel}} A_1|| \ldots ||A_n$. Por el teorema [4.3.1] sabemos que $A \simeq^{P^*,P'^*} A'$.

Luego por el teorema [3.2.2] sabemos que $A \models_{P^*} \phi \iff A' \models_{P'^*} \phi$ como queríamos demostrar. □

Corolario 4.3.2 *El problema de alcanzabilidad sobre el* observador A_0 *para $A_0||_{\equiv_{Rel}} A_1|| \ldots ||A_n$ es equivalente al de $A_0||A_1|| \ldots ||A_n$.*

Demostración: La demostración es trivial ya que alcanzabilidad puede ser escrita en $TCTL$. □

Este corolario nos permite utilizar el autómata compuesto reducido, en lugar del compuesto de la manera estándar, para la verificación de propiedades.

4.4 Algoritmos

Luego de formalizar nuestro esquema de reducción, estamos en condiciones de presentar algoritmos que lo implementan. En este capítulo presentaremos dos algoritmos:

El primero se utiliza para componer los autómatas y es similar al algoritmo estándar con la diferencia que utiliza una función de relevancia para cocientar al mismo tiempo que compone.

El segundo algoritmo calcula una función de relevancia que cumple con sus postulados.

4.4.1 Algoritmo para calcular la composición utilizando *Rel*

El siguiente algoritmo realiza la composición de los autómatas junto con las cocientización utilizando una función de relevancia *Rel*. Esta debe cumplir con los requisitos descriptos anteriormente, obteniendo un autómata equivalente al definido en la sección anterior (ver definición 4.2), pero solo con sus locaciones alcanzables. Observar que el algoritmo también podría ser utilizado para componer sin reducción utilizando una función de relevancia que asocie a cada posición del observador todas los componentes.

Componer$_{Rel}(A_0, A_1, ..., A_n, Rel)$ retorna $< S_r, X_r, A_r, I_r, s_{0_r} >$

Para cada $i = 1..n$

$$\pi_i(s_{0_r}) = \begin{cases} \perp \text{ si } Ai \notin Rel(Init(A_0)) \\ Init(A_i) \text{ en caso contrario} \end{cases}$$

$S_r := \{s_{0_r}\}$
$AProcesar := \{s_{0_r}\}$
$A_r := \emptyset$
Mientras $\exists s \in AProcesar$
Tomar s
$S_r := S_r \cup \{s\}$
$T :=$ TransHabilitadas$(s, A_0, A_1, ..., A_n)$
Para cada $t \in T$
Construir $t_r :=< s, label(t), \psi_r, \alpha_r, s' >$ donde
$\psi_r = \bigwedge_{\{i/\pi_i(t) \neq Null\}} Guard(\pi_i(t))$
$\alpha_r = \bigcup_{\{i/\pi_i(t) \neq Null\}} Reset(\pi_i(t))$

$$\pi_i(s') = \begin{cases} \perp \text{ si } Ai \notin Rel(\pi_{i+1}(s')) \\ Dest(\pi_i(t)) \text{ si } \pi_i(t) \neq Null \\ \pi_i(s) \text{ si } \pi_i(t) = Null \end{cases}$$

Si $t_r \notin A$
$A_r := A_r \cup \{t_r\}$
Si $s' \notin AProcesar$
$AProcesar := AProcesar \cup \{s'\}$
$S_r := S_r \cup \{s'\}$
Fin Para
$I_r(s) := \bigwedge_{\{i:1..n+1/\pi_i(s) \neq \perp\}} Inv(A_i)(\pi_i(s))$
$X_r := \bigcup_{i=1}^{n+1} Clocks(A_i)$
Fin Componer

TransHabilitadas$(s, A_0, \ldots, A_n)$
retorna $T =$Conj. de $< label, (n+1)$-tupla de $(Trans \cup \{Null\}) >$
$T := \emptyset$
$L := \bigcup_{i=0}^{n} \{Label(t)/t \in Trans(A_i) \wedge source(t) = \pi_i(s)\}$
Mientras $\exists e \in L$
Sacar e
$T_i(e, s) := \{t/t \in Trans(A_i) \wedge Label(t) = (e) \wedge source(t) = \pi_i(s)\}$
Si $(\forall i : 0..n)(\pi_i(s) \neq \perp \wedge e \in Labels(A_i) \Rightarrow T_i(e, s) \neq \emptyset) \wedge$
$(\nexists i : 0..n)(\pi_i(s) = \perp \wedge e \in Output(A_i))$ entonces
$T := T \cup \{< e, t_1, \ldots, t_n > /$
$(\forall i : 0..n \Rightarrow (t_i \in T_i(e, s)) \vee (T_i(e, s) = \emptyset \wedge t_i = Null))\}$
Fin Mientras
Fin TransHabilitadas

Correctitud y Orden del Algoritmo

El algoritmo recibe una función de relevancia que satisface sus postulados y construye la parte alcanzable (desde la locación inicial) del autómata cociente. Se utiliza un esquema simple de recorrido de grafos manteniendo los nodos ya procesados y los que aún no lo fueron. Este esquema es muy similar al algoritmo que se utilizaría para construir la composición estándar.

En nuestro caso, los nodos representan las clases de equivalencia, estas son tuplas con valores $\perp$ en las componentes no relevantes. Para cada nodo se calculan las transiciones habilitadas, verificando para cada etiqueta si la misma esta realmente disponible (ver definición de TransHabilitadas).

Asumamos que (a) las componentes proveen una lista de transiciones ordenadas por sus etiquetas en O(1), y (b) el algoritmo explora las etiquetas siguiendo ese orden. Entonces, el algoritmo ejecuta $O(V+E)$ pasos donde V es el número de nodos (locaciones) y E es el número de arcos (transiciones) en el grafo cociente. Esto es debido a que cada locación y transición son generados una sola vez y el procedimiento TransHabilitadas solo considera el conjunto de transiciones que comparten la misma etiqueta. Asumimos que este procedimiento puede ser implementado para que realice un apareamiento de las transiciones aprovechando la característica de que las mismas están ordenadas por sus etiquetas.

4.4.2 Algoritmo para calcular *Rel*

A continuación presentaremos un algoritmo para calcular *Rel*. Este calcula una función que cumple con sus postulados (ver definición 4.1.3) por medio de la técnica de mínimo punto fijo. Para el cálculo de influencia sobreestimamos los conjuntos $S_q(A_k)$ calculando la alcanzabilidad sobre las locaciones del autómata compuesto en vez de la alcanzabilidad sobre el LTS del mismo.

CalculaRel($A_0, A_1, ..., A_n$) retorna *Rel*

// Paso 1)Inicializamos Rel para cumplir con el ítem 1 de su definicion
Para cada $q \in Loc(A_0)$
    $Rel(q) := \{A_0\}$
Fin Para

// Paso 2.1 y 2.2. Actualizamos Rel para cumplir con el ítem 2 y 3 de su definicion
// Debemos repetir este proceso hasta que Rel se estabilice
Repetir
    // Paso 2.1
    $RelOld := Rel$

```
        Para cada t ∈ Trans(A_0)
            Rel(source(t)) := Rel(source(t)) ∪ Rel(dest(t))
        Fin Para
        // Paso 2.2
        Para cada q ∈ Loc(A_0)
            Para cada A_i/i : 0..n
                Para cada k/A_k ∈ Rel(q)
                    // Verifica influencia
                    Si VerificaInfluencia(A_0, ..., A_n, i, k, q)
                        Rel(q) := Rel(q) ∪ {A_i}
                Fin Para
            Fin Para
        Fin Para
    Hasta RelOld = Rel

Fin CalculaRel

VerificaInfluencia(A_0, ..., A_n, i, k, q) retorna resp
    S_{k_q} := {π_k(s)/s ∈ Loc(A_0|...|A_n) ∧ π_0(s) = q}
    I_{qk} :={Label(i) : i ∈ Trans(A_k)/source(i) ∈ S_{k_q}
        ∧(source(i) ≠ dest(i) ∨ (source(i) = dest(i) ∧ Reset(i) ≠ ∅))}
    O_i := Output(A_i)
    resp := (O_i ∩ I_{qk} ≠ ∅)
Fin VerificaInfluencia
```

Correctitud y Orden del Algoritmo

El algoritmo efectúa el clásico procedimiento para calcular un mínimo punto fijo que satisfaga los postulados de *Rel*.

Asumamos que el algoritmo es alimentado con la composición paralela estándar. Utilizaremos este dato para realizar una buena estimación de los conjuntos $S_q(A_i)$ (la llamaremos $S'_q(A_i)$). En realidad solo necesitamos las locaciones alcanzables del autómata compuesto que, por supuesto, ocupan mucho menos espacio que el autómata completo. El tamaño y costo del cálculo de esta entrada puede parecer muy grande pero en realidad no lo es ya que, debido a que el costo de la verificación es $PSPACE$ con respecto al tamaño, cantidad de relojes y valores del sistema, en la práctica sólo se trabaja con módulos de unas miles de locaciones (en el caso no temporizado suelen ser mucho más grandes) y como vimos en el algoritmo anterior el costo de su construcción es polinomial. Por lo tanto no es absurdo utilizar al producto completo como entrada de un algoritmo que será utilizado para

generar otro autómata en el que se reducirá de forma considerable los tiempos de verificación que son el verdadero problema a atacar.

Lema 4.4.1 *La complejidad del procedimiento para calcular Rel es polinomial con respecto al número de componentes (n), el número de locaciones de la composición estándar (V), la cantidad de locaciones de A0 ($m = \#Loc(A_0)$) y el máximo número de transiciones de los componentes individuales (T).*

Demostración: La sobreestimación de $S_q(A_i)$ para todo $q \in Loc(A_0)$ y $1 \leq i \leq n$, puede ser calculada recorriendo una sola vez la composición paralela en $n.V$ pasos (asumiendo que accedemos directo a la memoria para almacenar cada locación l de cada componente A_i y cada locación q de A_0 si $l \in S'_q(A_i)$. Esto podemos hacerlo utilizando una matriz de dimensión $[m, n]$ de arreglos booleanos de dimensión #Locaciones(A_i)).

En cada paso del cálculo del punto fijo, el nuevo *Rel* incluye el conjunto *Rel* previamente calculado e incorpora al menos un componente más en una locación. Por lo tanto, este puede ser iterado a lo sumo $m.n$ veces ($RelOld \subseteq Rel$). Asumamos que utilizamos programación dinámica para conservar si $A_i \xrightarrow{inf}_{S^*_q(A_k)} A_k$ o no. Para este análisis podemos asumir que esto no toma tiempo, luego calcularemos la complejidad de la obtención de todos estos valores.

Luego, en cada paso k, para cada locación q, el algoritmo chequea si los componentes no detectados relevantes influencian los ya detectados. Esto toma a lo sumo n^2 pasos (utilizando programación dinámica).

El paso 2.1 toma a lo sumo $T.n$ pasos. La comparación entre *Rel* y *RelOld* puede ser realizada en m pasos (el chequeo se puede implementar comparando la cardinalidad de cada locación dado).

Por lo tanto, sin asumir el costo del chequeo de influencia, el algoritmo toma a lo sumo $m.n.(m.n^2 + T.n + m)$ pasos, o sea (simplificando) un costo $O(m^2.n^3)$.

Por otro lado, existen al menos $m.n^2$ relaciones de influencia que deben ser realmente chequeadas. Para realizar ese chequeo, supongamos que los componentes proveen una lista ordenada por etiquetas de transiciones en $O(1)$. Entonces, dado el conjunto de locaciones $S'_q(A_i)$ y $S'_q(A_j)$ la lista ordenada de etiquetas asociadas a sus locaciones que usamos en la definición de influencia puede ser construida e intersectada en $O(T)$. Luego asumiendo el costo total del cálculo de influencia es a lo sumo $O(m.n^2.T)$.

Finalmente, tomando en cuenta todos los pasos mencionados y eliminando los ordenes no relevantes, podemos concluir que la complejidad del

algoritmo es $O(n.V + m^2.n^3 + m.n^2.T)$ □

4.5 Mejoras al método

Luego de realizar algunos experimentos pudimos concluir que nuestro método es efectivo. Sin embargo durante la experimentación descubrimos algunas mejoras que pueden realizarse al método. Algunas de ellas tienen impacto en la teoría por lo cual debemos adaptarla un poco.

Una mejora que se hizo evidente es que la definición de influencia era demasiada conservativa con respecto al componente influenciante. Si observamos la definición, notaremos que cualquier componente que exporte un Output que afecte a un autómata relevante, también es relevante. En la experimentación verificamos que esto no es realmente cierto, ya que puede suceder que la componente no tenga habilitada al Output influenciante para la locación actual del observador. Es decir, en realidad esa componente no podría realizar ese Output y por lo tanto no es relevante.

4.5.1 Teoría

Relevancia

Modificaremos la definición de influencia para que tome en cuenta cuando una transición de Output está disponible o no con respecto a una locación particular del observador.

A continuación presentamos la nueva noción de influencia.

Definición 4.5.1 (Influencia) *Dada una componente I/O A y un TA B*

$A \overset{inf}{\rightarrow}_{S1,S2} B$ *con* $S1 \subseteq Loc(B), S2 \subseteq Loc(B)$ *si* $O(A) \cap I(B) \neq \emptyset$

con $O(A) = \{o : Output(A) | \exists t \in Trans(A) \wedge source(t) \in S1\}$

y $I(B) = \{Label(t) : t \in Trans(B) | source(t) \in S2 \wedge (source(t) \neq dest(t) \vee reset(t) \neq \emptyset)\}$

Observar que la única diferencia con respecto a la definición original de influencia es que acotamos el conjunto de Outputs posibles para A.

Con respecto a los postulados de relevancia debemos realizar el siguiente cambio en el postulado número 3:

3. $k \in Rel(q) \wedge A_i \overset{inf}{\rightarrow}_{S_q^*(A_i), S_q^*(A_k)} A_k$

De esta manera la componente A_i influenciará al autómata A_k solo si la primera posee un Output que afecta al segundo pero además está disponible en el rango de locaciones $S_q^*(A_i)$.

Está modificación tiene como consecuencia que la función de relevancia sea más fina que la anterior.

La definición del autómata reducido es igual a la que realizamos previamente. Sin embargo, la restricción $(\nexists i : 0..n)(\pi_i(s) = \perp \wedge e \in Output(A_i))$ en la selección de transiciones tiene un mayor impacto, ya que antes eliminaba solo transiciones de Input que eran stutters. Con la nueva definición de influencia, se eliminan transiciones de Inputs no stutter pero que se quedaron sin su respectivo Output. Esto es muy importante ya que antes está restricción solo era una reducción "estética" porque eliminaba transiciones que no impactaban en el comportamiento del nuevo autómata y ahora es fundamental, ya que si no se eliminan estas transiciones el autómata resultante cambiaría su comportamiento (dejando Inputs no asociados a Outputs, que podrían ejecutarse cuando no deberían hacerlo).

Modificación del teorema de equivalencia

El teorema [4.3.1] sigue cumpliéndose con esta modificación y solo hay que hacer algunas aclaraciones. Los únicos puntos a aclarar serían las referencias al caso que un componente tenga un Output irrelevante. En donde mostramos que A' puede simular a A , hay una nota que dice que, en ese caso, las Inputs que sincronizan no se incluirían por la definición del autómata reducido y además serían stutters. Cabe aclarar que en el caso de que la componente que tiene el Output sea irrelevante por no estar disponible para la locación del observador, esta transición tampoco podría haberse realizado en A. Otra aclaración cabría en donde mostramos que A puede simular a A'. En la parte de la transición temporal dice que si a_{i_j} es una transición de Output irrelevante, esta sincroniza necesariamente con Inputs que hacen stutter. Aquí podemos aclarar que esto es así, ya que si el Output no sería relevante por no estar disponible para la locación del observador, simplemente no formaría parte de la corrida.

4.5.2 Algoritmos

El algoritmo de composición por *Rel* no necesita cambios. A pesar de ello, la reducción que este algoritmo logrará será mayor, ya que ahora podrá eliminar los Inputs no stutters que estaban asociados a Outputs irrelevantes.

La modificación al algoritmo de Cálculo de *Rel* es solo en el cálculo de influencia y es la siguiente:

VerificaInfluencia$(A_0, \ldots, A_n, i, k, q)$ retorna $resp$
$S_{k_q} := \{\pi_k(s)/s \in Loc(A_0||\ldots||A_n) \wedge \pi_0(s) = q\}$
$S_{i_q} := \{\pi_i(s)/s \in Loc(A_0||\ldots||A_n) \wedge \pi_0(s) = q\}$
$O_{qi} := \{Label(o) : o \in Trans(A_i)/Label(o) \in Output(A_i) \wedge source(o) \in S_{i_q}\}$
$I_{qk} := \{Label(i) : i \in Trans(A_k)/source(i) \in S_{k_q}$
$\wedge(source(i) \neq dest(i) \vee (source(i) = dest(i) \wedge Reset(i) \neq \emptyset))\}$
$resp := (O_{qi} \cap I_{qk} \neq \emptyset)$
Fin VerificaInfluencia

La modificación del algoritmo no tiene impacto en el orden del mismo ya que el costo del cálculo de de O_{qi} es similar al de I_{qk}.

Capítulo 5

Ejemplos

Luego de desarrollar la teoría y los algoritmos ha llegado el momento de evaluar la efectividad y propiedades de nuestro método. Para el desarrollo y elección de los ejemplos nos realizamos las siguientes preguntas:

1. ¿Qué impacto tiene la arquitectura de las componentes en la eficacia del método?
    - Acoplamiento entre componentes.
    - Aspectos temporales (valores de la constantes en las guardas de invariantes y transiciones).
2. ¿Cómo afecta la topología del Observador a nuestro método?
    - ¿Más locaciones implica mayor reducción?
    - ¿Cúal es el impacto del tamaño del Observador en el tamaño final del autómata compuesto?
3. ¿Es aplicable a casos reales?
4. ¿La aplicación del método permite la verificación de problemas que antes eran intratables?
5. ¿Que debilidades tiene el método?
6. ¿Que aspectos pueden mejorarse?

Para analizar los ejemplos es importante aclarar que es para nosotros la noción de acoplamiento. Para graficar esta noción veamos al sistema como un digrafo en el que los componentes representan los nodos y la influencia entre los componentes representan los arcos. Bajo esta mirada, diremos

que el grado de acoplamiento del sistema se corresponde con el tamaño de las componentes fuertemente conexas. Cuando mayor sea el tamaño de estas componentes mayor será el acoplamiento. Por ejemplo, un sistema asíncrono en el cual la arquitectura del sistema se asemeje a un pipeline tendrá un bajo acoplamiento. Por el contrario un sistema con una arquitectura en la cual sus componentes aparecen más entrelazados, tendrá un alto acoplamiento.

Para responder a las preguntas que nos hemos realizado, implementamos nuestros de algoritmos de composición y cálculo relevancia (ver Anexo A y CD adjunto con este trabajo). Cada ejemplo está conformado por varios componentes y varios observadores. En los experimentos, realizamos la composición de los módulos del sistema junto con uno de los observadores y realizamos una consulta de alcanzabalidad sobre el sistema de alguna proposición *Error* que se encuentra en el observador.

Nuestras hipótesis son las siguientes:

- El método siempre reduce, al menos para los estados trampa del Observador.
- Cuando menor es el acoplamiento del sistema mayor es la reducción.
- El tiempo de verificación en los autómatas reducidos puede ser significativamente menor y hacer tratables casos que antes no lo eran.
- A pesar de que ciertas propiedades pueden describirse con muy pocas locaciones, en general es conveniente utilizar Observadores con varias locaciones, utilizando un mayor detalle en las consultas, para permitir la reducción en los estados intermedios. Creemos que esto no afecta el tamaño final del autómata compuesto y mejora la eficacia del método.

Como primer paso aplicamos nuestra herramienta sobre los ejemplos y registramos los tamaños de los autómatas compuestos de manera estándar y el reducido. Utilizando KRONOS como herramienta de verificación, calculamos los tiempos de verificación de la consulta de alcanzabilidad para ambos autómatas resultantes.

Para cada caso de estudio utilizaremos dos tipos observadores (ver por ejemplo la figura 5.3): el primero será más "simple" en el sentido de que contendrá una secuencia de eventos que nos llevarían a la locación trampa con la proposición de error y en caso de que no se haya detectado esa condición volverá a la locación de inicio. El segundo observador será más adecuado para el uso con nuestra herramienta. En este caso, cuando no se detecte esa condición se irá una locación trampa que indica que no se produce esa condición, evitando el ciclo a la locación de inicio. Este detalle es, para nosotros,

de vital importancia ya que de esta manera se evita que el observador tenga (debido al postulado 2 de la definición de relevancia) baja reducción por relevancia. Notar que *Rel* tiene la misma imagen para todas las locaciones del observador de una misma componente conexa.

Queremos constatar con estos ejemplos que no perdemos eficiencia usando el segundo tipo de observador (conviente para nuestro método) con respecto al primer tipo de observador que, en principio, tiene menos locaciones. En algunos casos también se intentará utilizar un mayor nivel de detalle en cuanto a los eventos que nos llevan a la locación de error para verificar si esto ayuda a obtener una mejor reducción.

Para cada observador realizaremos una verificación "backward"(basada en el análisis de los predecesores, ver [DOTY96]) sobre un caso donde el error es alcanzable y sobre otro que no, y una verificación "forward"(sobre el grafo de estados alcanzables) para el caso donde el error es alcanzable (no lo haremos para el otro caso porque en general, para estos casos, está técnica suele ser muy costosa en tiempo). Para producir los casos negativos y positivos, variamos (sobre cada observador) el valor de las constantes asociadas a las condiciones sobre los relojes.

5.1 Net - Respuesta acotada en el sistema Mine Drainage

Comenzaremos nuestra experimentación con un ejemplo bien conocido basado en un caso real: El diseño de un controlador para un sistema de extracción de agua de una mina. Este ejemplo es comúnmente referenciado en la literatura (por ejemplo en [Jos96, BW96]) y tiene muchas de las características que poseen los sistemas de tiempo real embebidos. La versión presentada aquí es una extensión de [BW96] también presentada en [Bra00] y utiliza las técnicas de consulta basadas en observadores utilizadas en [BF99]. En la figura 5.1 podemos observar el diseño del sistema con una notación similar a la utilizada en [BW95].

El sistema es utilizado para bombear agua de una mina que se colecta en la base para llevarla a la superficie. Para evitar el peligro de explosión, la bomba no debe ser operada cuando el nivel del gas metano en la mina alcanza un valor alto. Los valores del ambiente son leídos periódicamente (flujo de aire, monóxido de carbono, flujo de agua) menos el valor del nivel de agua (alto y bajo) que es comunicado vía interrupciones. El objeto protegido *Motor* provee los servicios para operar la bomba y observar el status del motor. El objeto protegido *CH4Status* conserva el valor de la última lectura del metano. Cuando hay una situación de riesgo (el nivel de gas o el flujo de aire se vuelven críticos, el nivel del flujo de agua leído no coincide con el status del motor, etc.) se informa una alarma al objeto protegido *Operator Console*, para ser eventualmente señalizada a un monitor remoto en donde está el operador. Las operaciones y lecturas son colocadas en el objeto *Log*. Existe una tara esporádica *Command* utilizada para atender las solicitudes del operador remoto. Estas solicitudes son: inspeccionar el status del motor, prender o apagar la bomba. Los sensores de CO y CH4 usan la técnica "desplazamiento periódico" para realizar las lecturas: solicitan una una lectura que debería estar disponible en el siguiente período (sino se produciría una alarma). Hay otro mecanismo de detección de fallas. Una tarea chequea periódicamente la disponibilidad del sensor del nivel de agua enviando primero una solicitud y luego leyendo los *acknowledges* recibidos y encolados por una tarea esporádica en el ciclo anterior.

En el procesador remoto, hay una tarea esporádica que atiende las señales de alarma de la Consola del Operador. El *Console Proxy*, que corre sobre el procesador central, periódicamente envía una señal para indicar que está disponible.

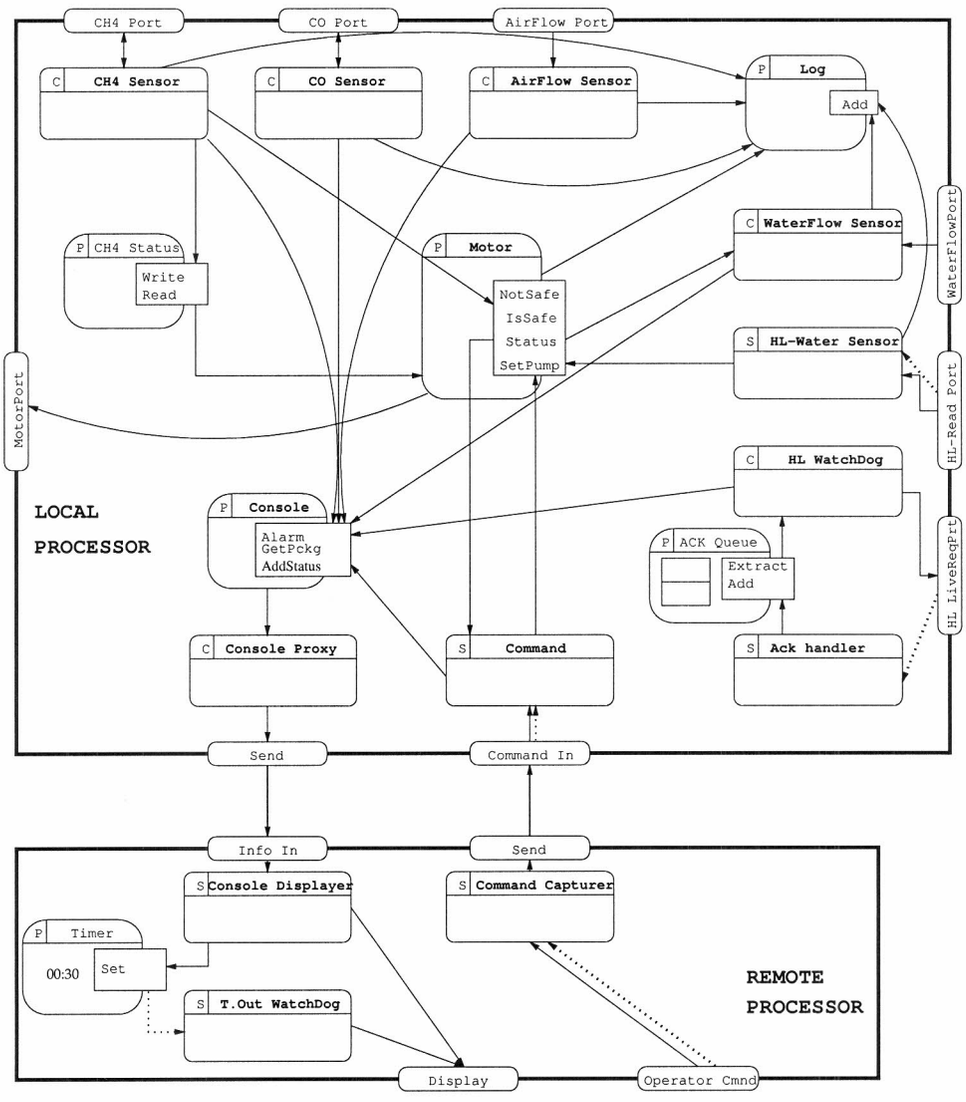


Figura 5.1: Diseño del sistema

A continuación veamos la modelización utilizando componentes temporizados (ver [BF99, Bra00] para conocer la técnica obtener el modelo en TAs a partir del diseño).

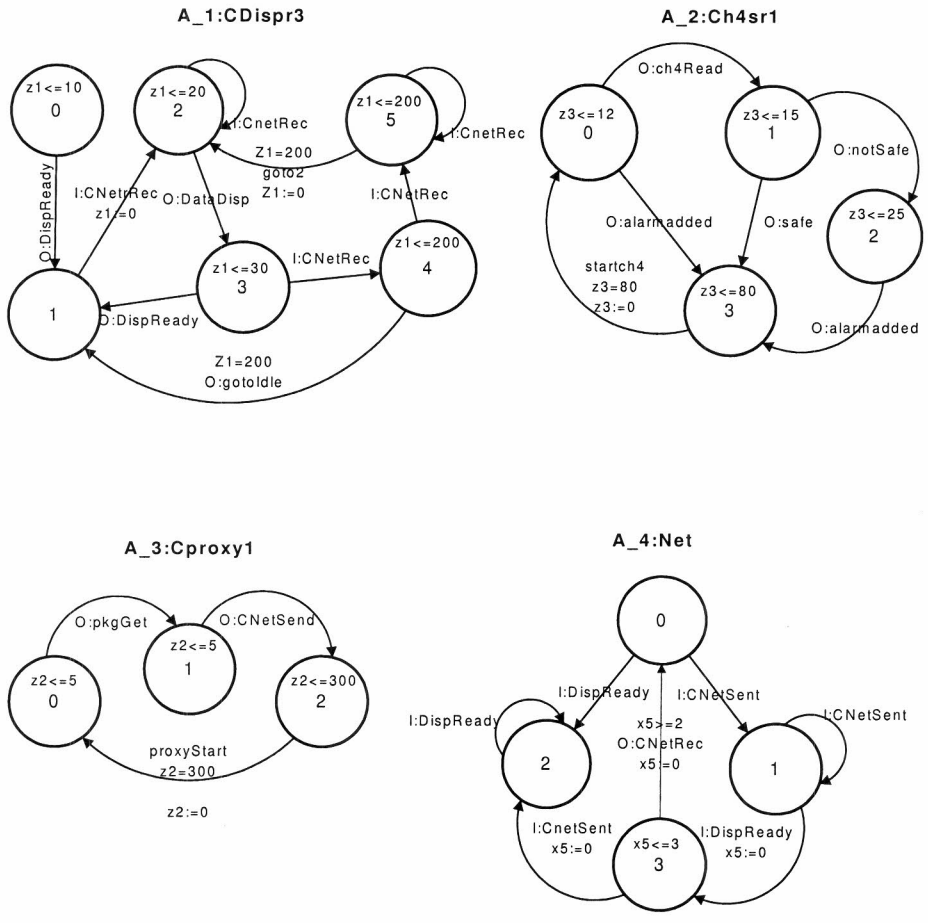


Figura 5.2: Net: Componentes del sistema

Lo que nos interesa saber es que si los valores leídos por el sensor de Ch4 son inseguros, las señales de alarma emitidas por el sensor, son leídas por el operador remoto en un tiempo menor a un valor *Top* (en este caso 1000). Se trata de una consulta de respuesta acotada.

Como vemos los componentes están poco acoplados. De alguna manera, el observador es el que le da la semántica correcta a los eventos que ocurren en las componentes. Por ello nuestro observador en su consulta debe "conectar" todos los eventos que deben suceder para que ocurra el comportamiento que deseamos. La secuencia de pasos a seguir es la siguiente:

- El sensor de CH4 lee el dato
- Si es inseguro, escribe en la consola que existe una situación de alarma
- El proxy lee la consola en el período correspondiente
- Se envía la alarma por la red
- Lo lee el console displayer y se lo muestra al operador

Deseamos que está secuencia se complete en un tiempo menor a un valor determinado.

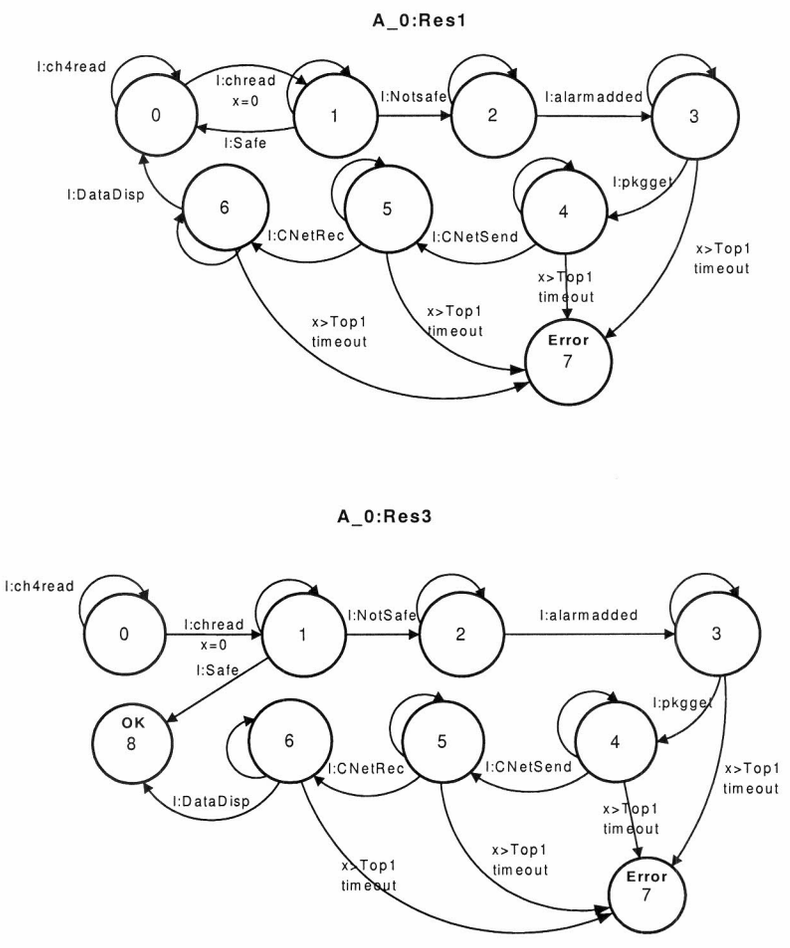


Figura 5.3: Net: Observador con ciclo del sistema y observador del sistema sin ciclos y con trampas.

A continuación los resultados del estudio:

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	658	2693	94.16s	143.02s	715e 1096t
Reducida	519	2203	97.40s	142.97s	715e 1096t

Tabla 5.1: Net: Composición Estándar vs. Reducida para el observador con ciclo

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	798	3183	94.17s	96.97s	715e 1057t
Reducida	267	963	1.64s	4.03s	262e 454t

Tabla 5.2: Net: Composición Estándar vs. Reducida para el observador con trampas

Como podemos observar, en el caso del observador con trampas la reducción obtenida con el método es realmente significativa. Esto se debe a que el sistema no es acoplado. Es más, el sistema se "conecta" a partir del observador que en la consulta conecta las acciones de las diferentes componentes. Por ello una vez que el observador no consulta más por algunos componentes, estos dejan de ser relevantes. Esto puede corroborarse al analizar el cálculo de relevancia. En el mismo se puede corroborar cómo el sensor de Ch4 deja de ser relevante luego de que el observador está entre las locaciones 2 y 3 y recibe su señal de alarma. Lo mismo sucede luego con el Console Proxy y la Red en las siguientes locaciones. Esto puede verse claramente en el análisis de relevancia. En la tabla 5.3 podemos ver un comparativo entre los cálculos de relevancia para los dos observadores.

Un punto a destacar es la diferencia en la eficacia del método con respecto a los dos observadores utilizados. Al utilizar el observador con trampa en OK logramos una reducción mayor. Esto se debe a que en el primer observador, los casos "OK" son devueltos a la locación inicial, por que , tal como lo indica la observación 4.3 de la definición de *Rel*, el ciclo genera que las componentes fuertemente conexas tengan la misma función de relevancia, por lo que es conveniente una construcción adecuada del observador que, tomando en cuenta esta característica, evite los ciclos innecesarios y utilize trampas para caracterizar los diferentes casos.

Obs. con Trampa		Obs. Simple	
Locación	**Relevantes**	**Locación**	**Relevantes**
0	A_0, A_1, A_2, A_3, A_4	0	A_0, A_1, A_2, A_3, A_4
1	A_0, A_1, A_2, A_3, A_4	1	A_0, A_1, A_2, A_3, A_4
2	A_0, A_1, A_2, A_3, A_4	2	A_0, A_1, A_2, A_3, A_4
3	A_0, A_1, A_3, A_4	3	A_0, A_1, A_2, A_3, A_4
4	A_0, A_1, A_3, A_4	4	A_0, A_1, A_2, A_3, A_4
5	A_0, A_1, A_4	5	A_0, A_1, A_2, A_3, A_4
6	A_0, A_1	6	A_0, A_1, A_2, A_3, A_4
7	A_0	7	A_0
8	A_0	8	no corresponde

Tabla 5.3: Net: Cálculo de relevancia para la composición con el observador con Trampa en OK y el observador con ciclo

5.2 Net Fault Detection

En este ejemplo analizaremos un mecanismo de detección de fallas del Mine Drainage. Una tarea de sistema remota envía señales a una tarea *CDispr3* que (entre otras tareas) resetea un *Timer*. Si el timer no es reseteado a tiempo, éste despertará a un *WatchDog* para que le informe al operador la falta de actividad del procesador remoto o la red. Notar que la red modela la falla por medio de una transición a una locación "trampa". En la figura 5.4 podemos observar los componentes del sistema tolerante a fallas.

Lo que nos interesa es verificar es que, una vez producida la falla en la Net, el *WatchDog* activa la alarma antes de 2000 u.t.

El observador (ver figura 5.5) consulta que luego del evento *Fail* y se produzca el evento *DisplayAlarm* en un tiempo menor a $Top1$ (en el caso real es 2000). Si se supera este valor se produce el Error.

En lo respectivo a nuestro método nos interesa saber que sucede con la relevancia de los componentes luego de que se produce la falla.

En las tablas 5.4 y 5.5 podemos ver los resultados del estudio. La primera conclusión obvia es que en este caso también es conveniente utilizar un observador sin ciclos. La diferencia entre ambos autómatas reducidos es realmente notable.

Como podemos observar la reducción es importante. La diferencia entre los tiempos del autómata compuesto de manera estándar y el reducido, tanto en la verificación backward como en la forward es realmente significativo. Esto es debido a que luego de que se produce la falla la red, los componentes Net y Cproxy dejan de ser relevantes (justamente debido a la falla) , por lo que además de reducir el tamaño drásticamente, desaparecen condiciones

sobre los relojes que aumentan considerablemente el costo del análisis.

En el análisis de relevancia (ver tabla 5.6), se observa claramente el motivo de la reducción.

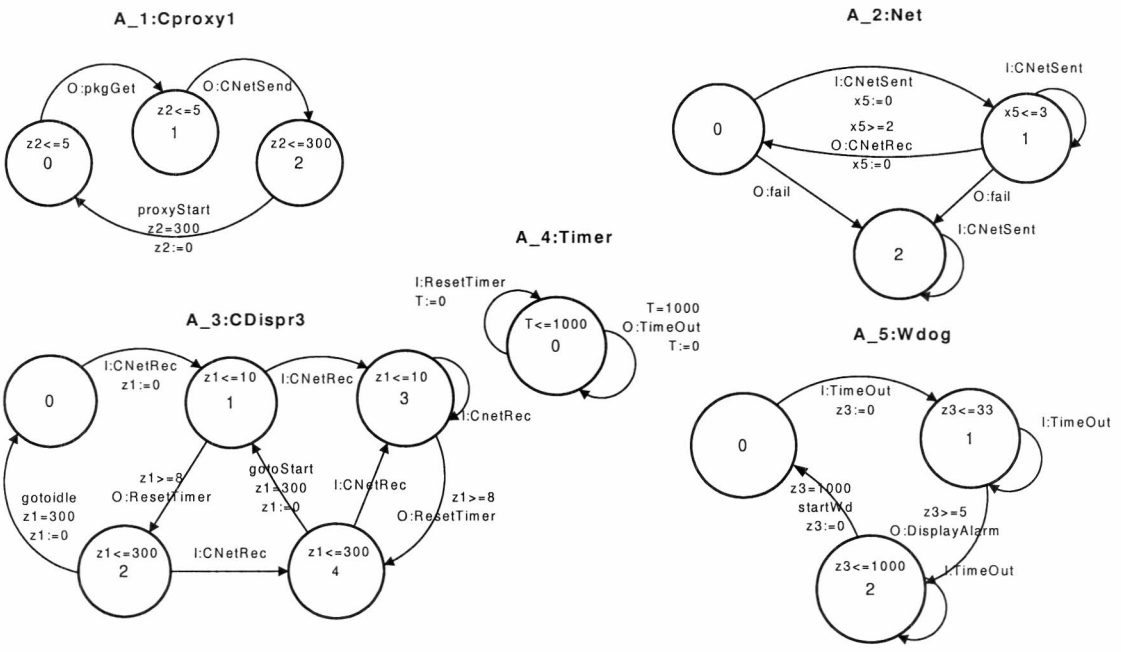


Figura 5.4: NetFd: Componentes de la red tolerante a fallas

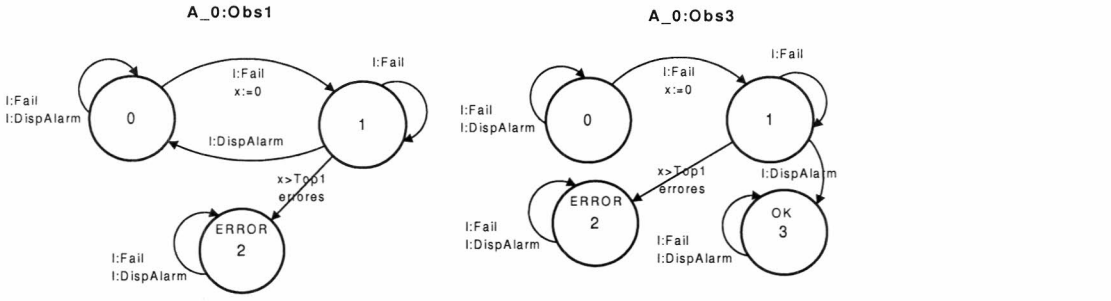


Figura 5.5: NetFd: Observador con ciclo y observador con trampas

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	225	1050	4634.91s	1410.35s	126e 132t
Reducida	181	894	4645.84s	1416.11s	126e 132t

Tabla 5.4: NetFd: Composición Estándar vs. Reducida para el observador con ciclo

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	270	1206	4632.36s	1412.04s	126e 132t
Reducida	152	745	0.72s	21.10s	6e 5t

Tabla 5.5: NetFd: Composición Estándar vs. Reducida para el observador con trampas

Locación	Relevantes
0	$A_0, A_1, A_2, A_3, A_4, A_5$
1	A_0, A_3, A_4, A_5
2	A_0
3	A_0

Tabla 5.6: Net:Fd Cálculo de relevancia para la composición con el observador con trampas

5.3 Struct - Sistema de Control estructural ante sismos

A continuación analizaremos otro interesante problema mencionado varias veces en la literatura: El sistema de control activo de estructuras [ECJ97, KLK96], etc. La versión de este ejemplo que estudiaremos aquí es una adaptación presentada en [Bra00].

Las estructuras activas incluyen un sistema embebido utilizado para limitar la vibración estructural debida a terremotos o vientos fuertes. Estas estructuras incluyen actuadores que pueden expandirse o contraerse para contrarrestar las fuerzas externas que son aplicadas sobre la estructura. Un controlador debe medir el estado de la estructura (i.e. aceleración y desplazamientos) y enviar comandos a los actuadores cuando las lecturas indican un estado no deseado. Para minimizar los daños producidos por las vibraciones inducidas por los terremotos, la frecuencia natural de la estructura

deber estar fuera de la banda de frecuencia de la excitación sísmica provocada por los terremotos. Un sistema de control censa las excitaciones sísmicas con una alta frecuencia de muestreo y cambia las frecuencias naturales de la estructura usando a los actuadores, todo esto de acuerdo a un algoritmo de control. Por supuesto, son necesarias ciertas restricciones temporales sobre la actividad de los actuadores debido a las cotas de tiempo requeridas por el algoritmo de control (si no son satisfechas, la estructura podría volverse inestable). La solución propuesta en [ECJ97] usa un algoritmo basado en un pulso de control. Para evitar que el desplazamiento vibratorio se acerque a la resonancia de la estructura, estos algoritmos aplican un pulso contrario a una frecuencia mayor para romper con las fuerzas resonantes. Las variables más importantes en el diseño son el tiempo entre iniciación de pulsos Δt_p, y la duración del pulso Δt. Estos valores están determinados por las frecuencias naturales del sistema, las funciones de fuerza esperadas y el nivel deseado de control de desplazamiento. La teoría tradicional de control de pulsos requiere que las demoras entre pulsos sean exactamente iguales, digamos, Δt_p. Sin embargo, asegurar que estas esperas son invariantes es casi imposible, donde juegan factores como el tiempo necesario para calcular las magnitudes del pulse y el tiempo de acomodamiento de los actuadores. Nosotros supondremos que la estructura se comporta correctamente si la espera entre pulsos es como mínimo 37 msec * 10^{-1} pero no más que 145 msec * 10^{-1} (a partir de aquí 10^{-1} significa una unidad de tiempo).

La siguiente tabla resume la información de tiempos para las diferentes actividades del sistema:

Actividad	Duración (en u.t.)
Muestro	[50,55]
Duración del Pulso	[25,30]

Tabla 5.7: Requerimientos Computacionales

Se utiliza una comunicación basada en sockets. Para estableces una conexión son necesarios 10 u.t., cuando ambos pares están listos para comunicarse toma 5 u.t. para intercambiar un mensaje y reconocimiento.

A diferencia del modelo presentado en [ECJ97], nosotros tratamos con una descripción de bajo nivel donde varias tareas interactúan. Por es, nuestra solución no es una maquina de estados monolítica como en ese trabajo. En realidad, varias tareas comparten un único procesador bajo una política de prioridad fija (ver figura 5.6). En particular, el algoritmo de control de pulso está mapeado sobre dos tareas, *modeler* y *pulser*.

El *modeler* es una tarea esporádica que, una vez inicializada, atiende mensajes que recibe desde el sensor. Luego, actualiza de forma predictiva

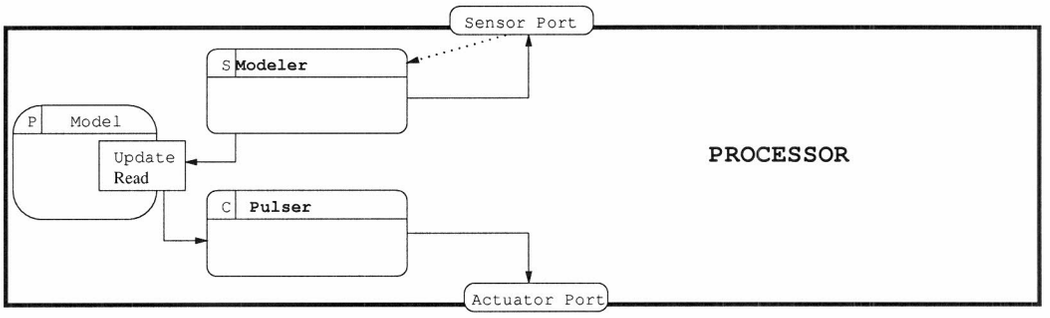


Figura 5.6: Arquitectura del sistema

el modelo que es almacenado en un objeto protegido compartido *Model* y comienza una nueva comunicación para ser atendida en la siguiente iteración. El *pulser* lee periódicamente el modelo, calcula la magnitud del pulse y comienza una comunicación con el actuador,no se queda esperando el reconocimiento del mensaje porque el diseñador asegura que este siempre llega antes que comience el siguiente período (110 u.t.).

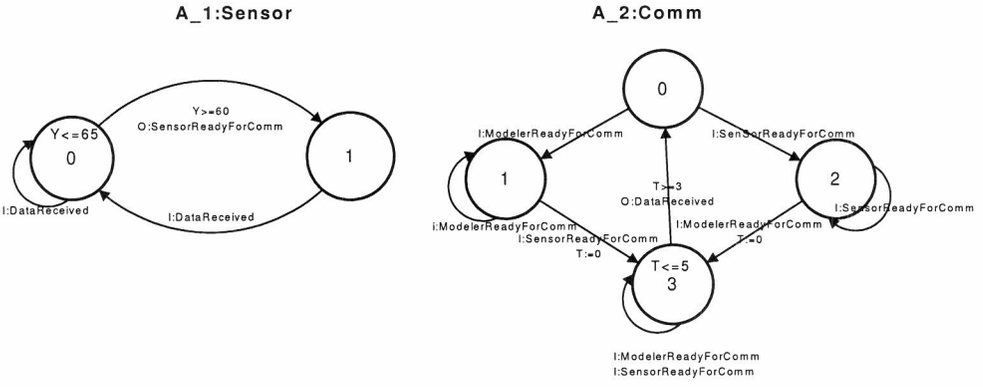


Figura 5.7: Struct: Componentes del sistema - Sensor y Comm

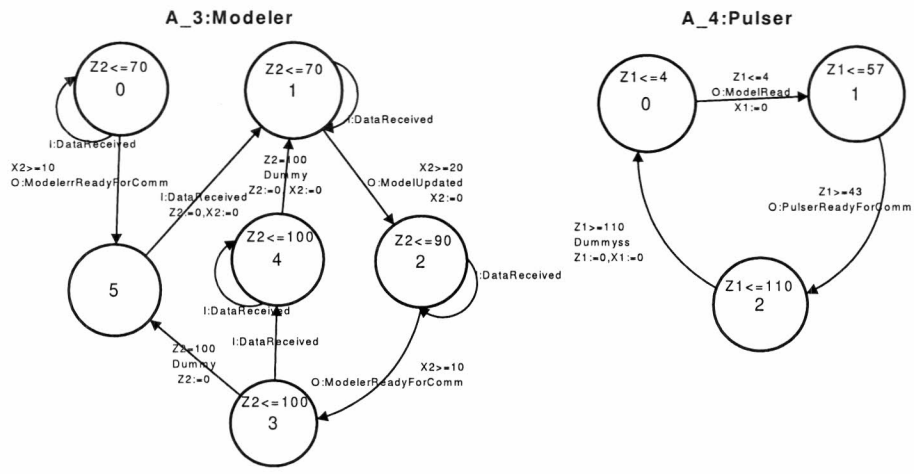


Figura 5.8: Struct: Componentes del sistema - Modeler y Pulser

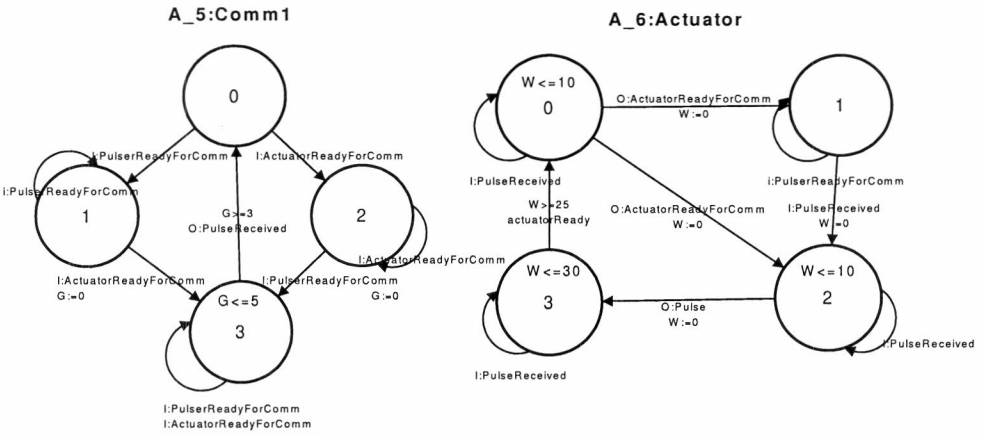


Figura 5.9: Struct: Componentes del sistema - Comm1 y Actuator

Existen dos requerimientos temporales principales para el sistema: frescura y regularidad: El primero indica que la validez del modelo actualizado por el *modeler* es a lo sumo 200 u.t. El segundo indica que la distancia temporal entre pulsos debe ser al menos 40 u.t. pero no más que 145 u.t.

A continuación aplicaremos nuestro método a los observadores correspondientes a estos dos requerimientos.

Frescura del modelo

Primero analizaremos la frescura del modelo. Al ejecutar el cálculo de relevancia para esta consulta (ver tabla 5.10) observamos que los componentes *comm1* y *actuator* no son relevantes para ninguna locación del observador ya que estos componentes no tienen que ver con la frescura del dato sino con el manejo de los pulso. Debido a esto existe una gran diferencia entre la composición estándar y la reducida y la verificación en el primer caso se hizo intratable. De todos modos, debemos destacar que detectar la no influencia de estos componentes no es trivial, ya que esta solo se hace evidente haciendo un análisis de los Inputs y Outputs de los mismos. Para poder estudiar mejor a nuestro método en nuestro caso lo comparamos también contra la composición estándar pero sin estos dos componentes (a cual llamamos Estándar 2).

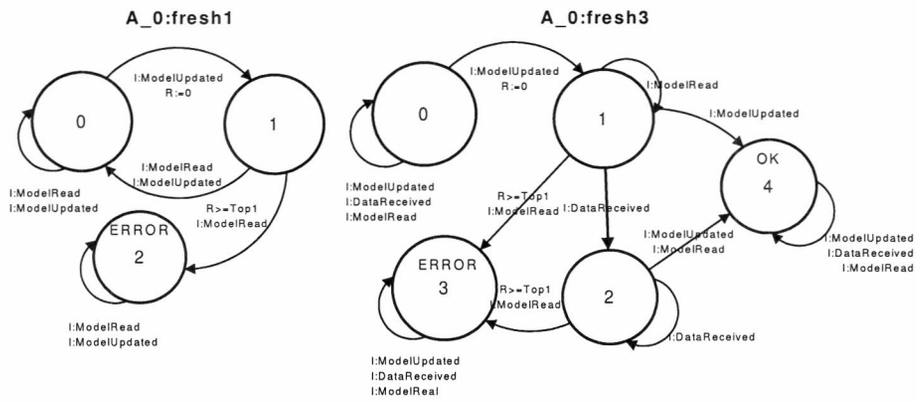


Figura 5.10: Fresh: Observador con ciclo y observador con trampas

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	768	2720	⊥	⊥	37391e 45013 t
Estándar 2	96	256	0.895s	28.52s	644e 738t
Reducida	67	181	0.87s	28.47s	644e 738t

Tabla 5.8: Fresh: Composición Estándar vs. Reducida para el observador con ciclo

Como podemos observar, comparando las tablas 5.8 y 5.9, la reducción utilizando el observador con trampas nuevamente es muy superior a la misma utilizando el observador con ciclo. Esto es debido a que el primero permite que en la locación 3 no sean más relevantes el *sensor* y el *comm*. Esto puede ser visto claramente en el análisis de relevancia (ver tabla 5.10). Allí

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	1008	3530	$\perp$	$\perp$	37391e 45013t
Estándar 2	126	331	0.90s	12.37s	644e 699t
Reducida	62	161	0.44s	11.28s	414e 459t

Tabla 5.9: Fresh: Composición Estándar vs. Reducida para el observador detallado con trampas

también podemos observar como las componentes *actuator* y *comm1* nunca son relevantes. Como mencionábamos anteriormente esta es la principal razón entre la diferencia entre la composición estándar y la reducida.

Sin embargo, los tiempos de verificación entre la composición estándar 2 y la composición reducida no son tan notorios como en otro casos. Esto se debe a que, a pesar de la reducción en el tamaño, los factores temporales (tamaño de constantes en las guardas e invariantes) que influyen sobre el costo de la verificación y no son controlados por nuestro método, tienen una influencia mayor en el costo de la verificación que el tamaño del automata en sí (locaciones y transiciones).

Locación	Relevantes
0	A_0, A_1, A_2, A_3, A_4
1	A_0, A_1, A_2, A_3, A_4
2	A_0, A_3, A_4
3	A_0
4	A_0

Tabla 5.10: Fresh: Cálculo de relevancia para la composición con el observador con trampas

Distancia entre pulsos

Ahora analizaremos la regularidad del modelo. En este caso el método detecta que los componentes *sensor* , *comm* y *modeler* no son relevantes para esta consulta. En este caso la no influencia de los componentes es más evidente ya que no participan en ninguna sincronización. Para este ejemplo mostraremos los resultados de la composición directamente sin estos componentes ya que el autómata que resulta de la composición de todos los componentes es intratable.

En la figura 5.11 podemos ver el observador con ciclo del sistema y el observador con trampas.

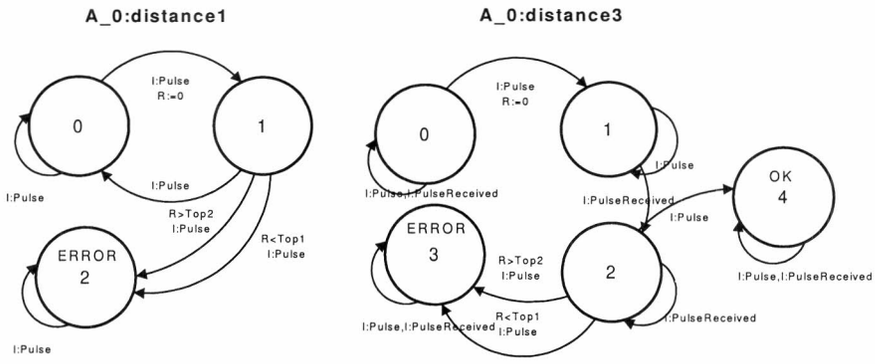


Figura 5.11: Distance: Observador con ciclo y observador detallado con trampas

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar 2	72	153	0.20s	0.09s	38e 38t
Reducida	49	108	0.19s	0.08s	38e 38t

Tabla 5.11: Distance: Composición Estándar vs. Reducida para el observador con ciclo

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar 2	96	198	0.07s	0.07s	38e 38t
Reducida	45	87	0.05s	0.05s	38e 38t

Tabla 5.12: Distance: Composición Estándar vs. Reducida para el observador con trampas

Los resultados para estudio son muy similares a los del ejemplo anterior (ver tablas 5.11 y 5.12). Con el observador con trampas se obtienen mejores resultados con respecto al tamaño del sistema ya que luego de la locación 2 el *pulser* y el *comm1* dejan de ser relevantes. Sin embargo los tiempos de verificación son similares debido a la probable incidencia de los aspectos temporales del sistema (constantes de los relojes en las guardas) que tienen mayor impacto en el costo de verificación que el tamaño del sistema (locaciones y transiciones).

Locación	Relevantes
0	A_0, A_4, A_4, A_6
1	A_0, A_4, A_5, A_6
2	A_0, A_6
3	A_0
4	A_0

Tabla 5.13: Distance: Cálculo de relevancia para la composición con el observador con trampas

Freshness sistema completo

Lo que nos interesa en este caso es saber que la edad del dato utilizado en los pulsos no supere cierto valor, es decir, lo que deseamos es verificar la frescura del dato y la consecuente emisión del pulso. Para realizar esta consulta, el observador debe involucrar a todos los componentes del sistema, por lo que el tamaño del automata compuesto será mucho mayor.

En este caso examinaremos solo los resultado correspondiente a la composición con observadores sin ciclos ya que en ese caso el tamaño del sistema es demasiado grande y es intratable para la herramienta $KRONOS$. Realizaremos el experimento utilizando dos observadores que realizan la consulta. El segundo tendrá un mayor nivel de detalle al primero (describirá una mayor cantidad de eventos para llegar al error).

En la figura 5.12 podemos ver los observadores del sistema. Estos primero verifican que, una vez que el dato utilizado para el cálculo del pulso, no sea más viejo que $Top1$ u.t. cuando el pulso es aplicado.

En las tablas 5.14 y 5.15 presentamos los resultados del estudio. Como podemos observar la reducción es realmente notable y, sobre un problema que no podía verificarse porque era intratable, al menos se logró verificar todos casos para la verificación forward y el caso de no alcanzabilidad para la backward. La razón para tan drástica reducción se debe a que en el cálculo de relevancia (ver tabla 5.16), el método descubre que a medida que vamos avanzado por las locaciones, ciertos componentes dejan de ser relevantes.

Es importante destacar que, aunque en ambos casos la reducción es notable, utilizando el observador con más detalle logramos un resultado aún mejor. Esto se debe a que cuando el observador tiene mayor nivel de detalle posee mayor información para el cálculo de influencia, por lo que es mayor la posibilidad de que encuentre componentes no influenciantes.

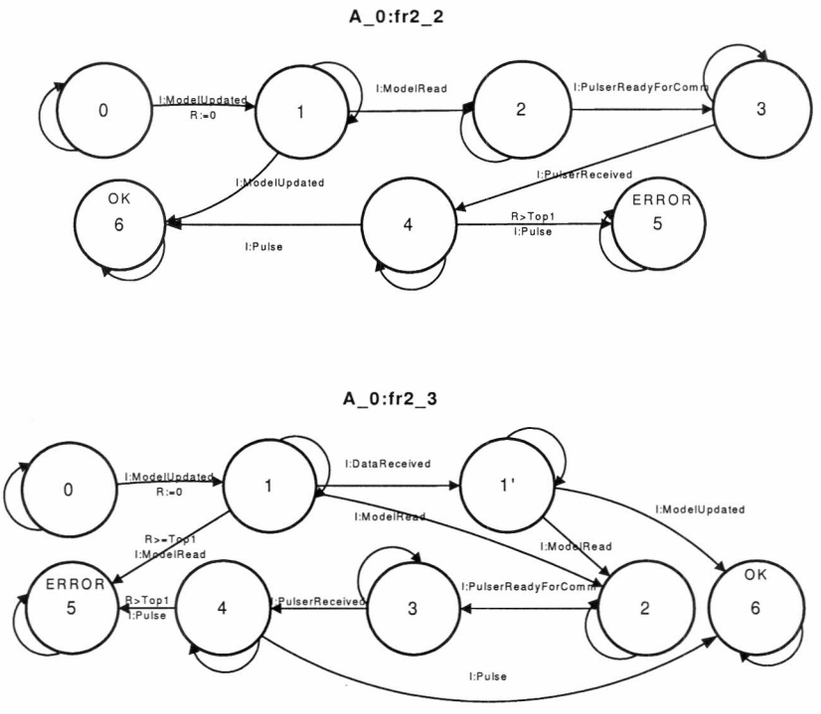


Figura 5.12: Freshness: Observadores del sistema

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	1268	4454	⊥	⊥	6670e 9253t
Reducida	543	1856	6341.31s	⊥	1478e 1890t

Tabla 5.14: Freshness: Composición Estándar vs. Reducida para el primer observador

Composición	Locs.	Trans.	Reach? False	Reach? True	
			Backward	Backward	Forward
Estándar	1268	4410	$\perp$	$\perp$	6670e 7943t
Reducida	495	1665	6140.94s	$\perp$	1478e 1820t

Tabla 5.15: Freshness: Composición Estándar vs. Reducida para el observador más detallado

Locación	Relevantes
0	$A_0, A_1, A_2, A_3, A_4, A_5, A_6$
1	$A_0, A_1, A_2, A_3, A_4, A_5, A_6$
1'	$A_0, A_1, , A_3, A_4, A_5, A_6$
2	A_0, A_4, A_5, A_6
3	A_0, A_5, A_6
4	A_0, A_6
5	A_0
6	A_0

Tabla 5.16: Freshness: Cálculo de relevancia para la composición con el observador más detallado

5.4 Train Gate Controller

A continuación utilizaremos nuestro método para comprobar su funcionamiento contra uno de los ejemplos más citados en la literatura (por ejemplo en [ACH+92, KLK96, DOTY96, TY96],etc.): El sistema Tren-Barrera-Controlador. Este ejemplo es utilizado frecuentemente tanto para ejemplificar la composición entre autómatas, como para ejemplificar métodos de reducción.

El ejemplo que presentaremos es un poco más complejo que los citados ya que contempla el paso de varios trenes (varios carriles) y el control de la barrera es un poco más complejo. A continuación presentaremos las componentes que componen al sistema. Las mismas fueron adaptadas a nuestro esquema de Input/Output compatibles. Como consecuencia de esto, nuestro modelo es más realista que los ejemplos presentados por otros autores ya que nuestro sistema es no bloqueante (no es realista suponer que el tren no avanza hasta que sincroniza con el controlador).

En la figura 5.13 prsentamos los componentes del sistema. El *Tren n* que emite 3 tipos de señales: *Approach n*, *In n* y *Exit n*. Cuando el *tren n* se aproxima a la barrera emite la señal *approach n* al *controlador* y entra al cruce al menos 300 u.t. después. Cuando el *tren n* se va del cruce, emite la señal *exit n* al *controlador* en un tiempo que debe ser de 500 u.t. luego de la emisión de la señal *approach n*. El *Controlador* emite una señal *Lower* a la

barrera exactamente 100 u.t. después de haber recibido la señal *approach n*, y emite la señal *raise* 100 u.t. después de recibir la señal *exit n*. La *barrera* responde a la señal *Lower* bajándola en 100 u.t. y responde a la señal *raise* subiéndola entre 100 y 200 u.t.

Lo que nos interesa saber es si es posible que un tren este pasando y la barrera no este completamente baja. En eso estará basada nuestra consulta. Para ello construiremos un observador (ver figura 5.14) donde se consultará la secuencia de los eventos *Raise* (comenzamos aquí para partir desde una barrera alta), *Lower* e *In* sin que haya un *Down* previamente, caso para el cual iremos a una locación de error. También se detecta que luego de un *Down* pase un tiempo antes de ocurra *In*, o sea el tren no puede pasar apenas se baja la barrera, sino que debe existir una pequeña demora. En el observador, para el caso en el que no se alcanza la condición de error, seteamos en 2 el valor de la demora. Para el caso en que el error es alcanzable seteamos la demora en 4 (también podría ser 3).

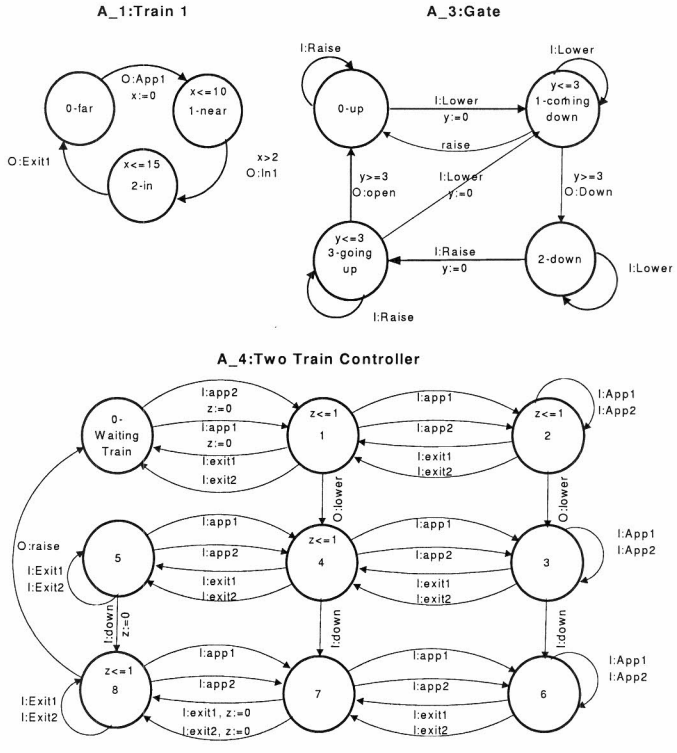


Figura 5.13: TGC: Sistema Tren-Barrera-Controlador para 2 trenes

El la tabla 5.17 presentamos los resultados del estudio para un sistema con cuatro trenes. En este caso los componentes A_1, A_2, A_3 y A_4 representan a los trenes, A_5 representa la barrera y A_6 el controlador. Como puede

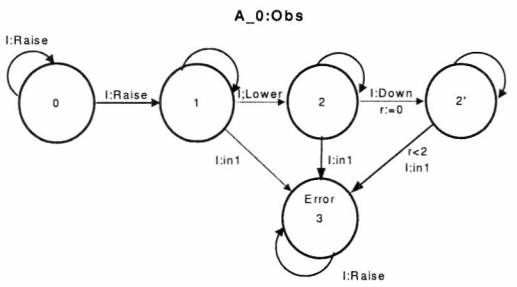


Figura 5.14: TGC: El observador que consulta sobre el tren 1

observarse, la reducción en el tamaño es realmente importante, también la diferencia en los tiempos de verificación en el caso en que la consulta de alcanzabilidad da falso (que es el caso que se ajusta al modelo real). Esto se debe a que al realizar el cálculo de relevancia (ver tabla 5.18), el método descubre que luego de la locación 2 el único tren relevante es el número 1 y el controlador ya no es relevante porque ya le indico a la barrera que debe bajarse (el controlar no podría influenciar más al sistema antes de que el tren no salga del paso a nivel, algo que sucede después del *In*. Es claro que luego de este evento no nos interesa el comportamiento de ningún componente ya que caemos en una locación trampa). En el caso en que la alcanzabilidad es verdadera, la reducción en los tiempos de verificación no es tan importante (con la constante elegida).

Composición	**Locs.**	**Trans.**	**Reach? False**	**Reach? True**	
			Backward	**Backward**	**Forward**
Estándar	1134	5828	39.66	165.97	26203e 30438t
Reducida	438	2220	1.38	156.45	26176e 30402t

Tabla 5.17: TGC: Composición Estándar vs. Reducida para el observador del Tren 1

Locación	**Relevantes**
0	$A_0, A_1, A_2, A_3, A_4, A_5, A_6$
1	$A_0, A_1, A_2, A_3, A_4, A_5, A_6$
2	A_0, A_1, A_5
2'	A_0, A_1
3	A_0

Tabla 5.18: TGC: Cálculo de relevancia para la composición con el observador del Tren 1

Sin embargo, a medida que aumentamos el valor de la demora mínima, el

tiempo de verificación backward para el autómata estándar comienza a aumentar mientras que en el autómata reducido el tiempo se mantiene estable. En el caso de la verificación forward, los valores se mantienen exactamente igual. En la tabla 5.19 podemos observar los tiempos de verificación para los distintos valores de las demoras. Como puede verse, a partir del valor 7 los tiempos en la composición estándar comienzan a dispararse. En cambio en la reducida se mantienen estables. Analizando en detalle ambos autómatas, verificamos que en el estándar existen 108 guardas en las que participa este reloj, en cambio en el reducido el mismo participa en una sola guarda. También notamos que este reloj aparece siempre ligado a la condición $T1 > 6$ perteneciente al reloj del tren 1. Esto explica porque a partir del valor 7 los tiempos comienzan a explotar.

Demora	**Estándar**	**Reducida**
3	186,92	158,36
4	165,97	156,45
5	177,04	163,33
6	176,42	159,73
7	823,57	159,86
8	1084,75	160,06
10	>2000	164,02

Tabla 5.19: TGC: Estudio del desarrollo de la demora para la verificación backward del Tren 1

5.5 Conclusiones sobre los ejemplos

A esta altura estamos en condiciones de responder a las preguntas que nos realizamos antes de realizar el estudio sobre los ejemplos.

1. ¿Qué impacto tiene la arquitectura de las componentes en la eficacia del método?

   El impacto es muy importante ya que pudimos verificar que cuando menor es el acoplamiento entre los componentes mayor es la posibilidad de reducción ya que el grado de influencia de los mismos es mucho menor. Cuando más acoplados son los componentes, menor es la efectividad del método. Esto significa que el método será más eficiente cuando la arquitectura del sistema se asemeje más a un DAG de componentes fuertemente conexas en el que la cantidad de nodos del mismo se acerque a la cantidad de componentes del sistema. De todos modos es notable la efectividad del método aún en casos globalmente acoplados como el TGC. En el mismo la estructura del observador hace que el acoplamiento local no sea tan importante.

Otro aspecto importante son los aspectos temporales intrínsecos en el sistema. En la mayoría de los ejemplos notamos que la reducción de locaciones y transiciones implicaba una reducción en el tiempo de verificación. Sin embargo, en algunos casos aislados, notamos que, a pesar de existir una importante diferencia en los tamaños del autómata estándar y el reducido, el tiempo de verificación fue prácticamente el mismo. Esto podría ser debido a que la complejidad del algoritmo de verificación depende tanto del tamaño del autómata como la cantidad de relojes y del valor de las constantes en las condiciones sobre los relojes. Podría suceder que, en algunos casos, la reducción no afecte a la condiciones que mayor impacto tienen sobre el costo de la verificación. De todos modos también verificamos que el autómata reducido, en general, tendría menor cantidad de apariciones de las guardas en donde aparecen esas condiciones, por lo que el tiempo de verificación sería menor.

2. ¿Cómo afecta la topología del Observador a nuestro método?

   La eficacia del método depende fuertemente de la topología del Observador. En general es conveniente que el mismo sea lo más detallado posible con respecto a la descripción de eventos. De esta forma, el analizador de relevancia poseerá más información y podremos ir desprendiéndonos de componentes no relevantes a medida que avanzamos las locaciones del observador.

   En general notamos que aumentar el tamaño del observador no tiene un gran impacto en el tamaño final del autómata compuesto (al menos en el reducido). Esto es debido a que las aristas que se colocan en el observador suelen sincronizar con otras componentes, evitando de esta forma, la explosión provocada por la combinación de locaciones. Además, como dijimos previamente, un observador con mayor detalle, tiende a mejorar la eficacia del método, por lo que el posible crecimiento del autómata estándar quedaría contrarrestado con la mayor reducción del autómata generado con nuestro método. Podemos concluir que utilizar componentes con trampas es mejor en todo sentido que utilizar componentes con ciclos.

   En la generación de los ejemplos, hemos generado observadores tendientes a verificar la posibilidad de ciertas condiciones de error. Hemos notado que el método es más eficiente justamente para los casos en que estas condiciones no se dan, o sea los casos en donde la alcanzabilidad es falsa. Esto significaría que podríamos utilizar nuestro esquema de reducción para verificar rápidamente que el sistema está libre de ciertas condiciones no deseables (es *safe*).

3. ¿Es aplicable a casos reales? ¿La aplicación del método permite la verificación de problemas que antes eran intratables?

En los ejemplos quedo claro que el método puede ser muy útil para el estudio de casos reales. En general pudimos observar que la utilización del método reduce drásticamente los tiempos de verificación, tanto en el caso backward como en el forward, en especial para el caso donde la consulta de alcanzabilidad es falsa. Más aún, existieron casos en los cuales no pudo realizarse la verificación para la composición estándar y sí para la composición reducida.

Un aspecto muy positivo del método, es que siempre reduce, al menos es las locaciones finales del observador (trampas). Además es una herramienta útil debido a que se pueden construir consultas y composiciones sin tomar en cuenta si realmente se utilizan todos los componentes, ya que el método los elimina automáticamente (ver, por ejemplo los casos de la consultas Frescura y Distancia en el ejemplo STRUT).

4. ¿Que debilidades tiene el método?

   En general el método cumple con su función: reducir el tamaño global del sistema eliminando componentes innecesarias con respecto a un observador. Quizás en algunos casos, pequeños o particulares, no es conveniente aplicarlo porque la suma del costo de calcular la función de relevancia más el de verificación del autómata reducida, puede ser prácticamente el mismo que el de la verificación del sistema original. En general este tipo de casos no requiere la utilización de esquemas reducción de ningún tipo.

5. ¿Que aspectos pueden mejorarse?

   Existen varios aspectos que pueden mejorarse. Los trataremos en el siguiente capítulo.

Capítulo 6

Conclusiones y Trabajo Futuro

6.1 Conclusiones

Definimos un método de reducción que puede ser muy útil cuando se deben utilizar autómatas de consulta para observar ciertas propiedades de un sistema. Hemos comprobado que nuestro esquema es en general efectivo para reducir el tamaños de los sistemas que son modelados de esta forma.

Las ventajas que consideramos que posee este esquema son las siguientes:

- Su utilización es muy simple. Se usa como una herramienta para componer autómatas. La única salvedad que hay que tomar en cuenta es clasificar las etiquetas para convertir a los autómatas en componentes Input/Output.
- Puede ser utilizado por personas que utilizan los sistemas temporizados como herramienta de modelización, sin tomar en cuenta, al momento de componer autómatas, cuáles son realmente necesarios o no, ya que el método los detecta.
- Construyendo autómatas de consulta (observadores) adecuados, el método siempre reduce. En general la reducción en el tiempo de verificación es muy significativa.
- La reducción en el tiempo de verificación es más notable cuando el error es no alcanzable (testeo de no alcanzabilidad).
- Permite la verificación de problemas que si no serían intratables.
- Es fácilmente integrable a herramientas de verificación.

- Permite reducir la cantidad de relojes de manera indirecta (al eliminar componentes, ver siguiente punto).
- Es ortogonal a otros métodos de reducción, por lo que un autómata generado por nuestro método puede ser sometido a otro esquema de reducción. Por ejemplo, luego de aplicarse el método, puede utilizarse la herramienta *Optikron* [DY96, Daw98] para detectar y eliminar los relojes que dejan de estar activos.

Con respecto al por qué de la mejor eficacia del esquema de reducción para la verificación cuando el error es no alcanzable, aún no logramos obtener una conclusión definitiva. Creemos que es debido a que, en estos casos, se alcanzarían más rápidamente los estados relacionados con el trampa de proposición OK y, como vimos en los ejemplos, en las locaciones trampa solo es relevante el observador. Justamente en el caso en el que el error es alcanzable sucedería lo contrario. El camino hacia el estado trampa Error es más largo y además, por la definición de influencia, necesariamente en los estados participantes de la verificación habría componentes comprometidos (por sincronizar con el observador), por lo que el espacio de estados sería mayor. Además, como hemos visto, la función de relevancia provoca que haya mayor de densidad de componentes cerca de los estados iniciales, densidad que va reduciéndose a medida que avanzamos por los estados del observador.

Consideramos que los puntos que no ataca el esquema o en los cuales no es eficaz son los siguientes:

- No es muy eficaz cuando el observador requiere un conjunto muy acoplado de componentes.
- Sólo ataca la reducción del tamaño del problema que es una parte importante del costo de la verificación, pero no ataca directamente los aspectos temporales del sistema (cantidad de relojes, el valor de las constantes en las condiciones) que es el otro factor importante en el costo de la misma.

Como conclusión final, creemos que la herramienta que hemos creado puede ser muy útil y es un buen complemento a las otras herramientas de verificación que atacan otro tipo de cuestiones.

6.2 Trabajo Futuro

Consideramos que luego de la realización de este trabajo quedaron algunos puntos en los que sería interesante profundizar.

A continuación proponemos una lista de temas en los cuales se podría continuar trabajando:

1. Integrar nuestro esquema de reducción a la herramienta KRONOS. El cálculo de relevancia brinda información sobre los componentes que quedan activos para cada locación del autómata observador. Integrando esta herramienta con *Optikron* [DY96, Daw98], esta información puede ser utilizada en la verificación para eliminar relojes innecesarios y reducir aún más el tiempo de reducción.

2. Mejoras en el cálculo de relevancia. La implementación de nuestro esquema, al realizar el cálculo de influencia entre componentes, no toma en cuenta la información temporal de los componentes. Esto es porque, a pesar de que la teoría utiliza los estados alcanzables del LTS del sistema compuesto para establecer la relación de los componentes, en la práctica sobreestimamos este cálculo calculando la alzanzabilidad sobre las locaciones del automata compuesto no temporizado. Claramente, esta sobreestimación puede llevarnos a determinar la influencia entre componentes que en realidad no lo son. Por lo tanto cuando más cerca estemos del LTS del sistema más fino será nuestro cálculo de influencia y por lo tanto también lo será el de relevancia.

3. Construcción automática de interfaces Input / Output. Sería interesante desarrollar una herramienta que pueda construir (o sugerir) las interfaces Input / Output de los componentes del sistema. Si esto se lograra, nuestro método de reducción sería totalmente automático.

4. Observadores fórmula dependientes. Como pudimos ver en el trabajo, nuestro método es *autómata de consulta* dependiente, basado en una consulta $TCTL$ de alcanzabilidad predicando sobre alguna proposición del mismo. Sería interesante hallar un método para poder convertir algún subconjunto de fórmulas $TCTL$ a autómatas de consulta. De esta forma podríamos obtener un método de reducción que sea fórmula dependiente. Ya que no estudiamos esta propuesta en detalle, no sabemos aún si la topología de los observadores resultantes sea la adecuada para ser utilizada sobre nuestro método. Tampoco estudiamos cuáles son las fórmulas que realmente pueden ser traducidas. Sin embargo, creemos que es interesante avanzar sobre esta línea de trabajo.

5. Heurísticas para mejorar la reducción. Por ejemplo si existe un componente que es siempre relevante de manera directa para todas locaciones del observador (menos en las trampa, en las que solo el observador es relevante), el mismo no sería reducido y potencialmente puede combinarse con los demás componentes. Una idea interesante sería armar un

nuevo observador a partir de la composición con nuestro método de este componente con el observador antiguo. De esta forma el tamaño de este componente relevante sería menor que en el caso original ya que el componente que antes no podía reducirse, ahora estaría reducido formando parte del nuevo observador. Además este observador sería más detallado (con las ventajas que esto podría tener) sin aumentar el tamaño global del sistema. Hay que tener en cuenta que esto sólo podría realizarse con una componente que es relevante en todas las locaciones, ya que en caso contrario no se cumplirían los postulados de la función de relevancia y podríamos obtener un autómata que no es observacional continuo bisimilar con el de la composición estándar.

Estos fueron algunos de los puntos que creemos que pueden ser desarrollados y esperamos que puedan surgir muchos más a partir como consecuencia del aporte que brinde este trabajo.

Apéndice A

Utilización de la Herramienta

La herramienta que presentamos implementa los algoritmos descriptos anteriores. Puede ser utilizada tanto para realizar la composición estándar como la reducida calculando automáticamente la función de relevancia correspondiente.

El modo de operación es el siguiente: se le debe pasar como parámetro un conjunto de componentes I/O, un componente I/O observador (ambos en formato *Kronos*) y el nombre que se desea darle al autómata compuesto, generando cuatro archivos de salida:

- El archivo *nombre*.tg es un archivo con formato *Kronos* con el autómata compuesto de la manera estándar.
- El archivo *nombre*.red.tg es un archivo con formato *Kronos* con el autómata compuesto cocientado según *Rel*.
- El archivo *nombre*.rel es un archivo con formato texto que describe la función de relevancia del observador.
- El archivo *nombre*.txt es un archivo con formato texto con el log de las operaciones realizadas.

Sintaxis:

$$\text{CalcuRel } nombre\ a0\ [a1...an]$$

donde *a0* es el observador del sistema, a1...an son los componentes del sistema y *nombre* es el nombre de los archivos de salida.

Formato de los autómatas:

El formato que acepta nuestra herramienta es el mismo que utiliza *Kronos*. La única diferencia es que a cada etiqueta se le debe especificar si es de Input, Output o interna agregando delante del nombre de la misma la frase *I:* u *O:* para los dos primeros casos o nada para el último. En la tabla A.1 podemos observar la codificación de los componentes del sistema presentado en la figura 4.2.

Observador:A0	**Componente:A1**	**Componente:A2**
#states 4 #trans 6 #clocks 1 X state: 0 invar: true trans: true => I:b; reset ; goto 0 true => I:a; reset X ; goto 1 state: 1 invar: X<10 trans: true => I:a; reset ; goto 1 true => I:b; reset X ; goto 2 x>=10 => FueMal ; reset X ; goto 3 state: 2 invar: true trans: state: 3 prop: Error invar: true trans:	#states 2 #trans 2 #clocks 1 Y state: 0 invar: true trans: true => O:a; reset Y ; goto 1 state: 1 invar: true trans: true => vuelta; reset ; goto 0	#states 3 #trans 6 #clocks 1 Z state: 0 invar: true trans: true => I:a; reset Z ; goto 1 state: 1 invar: true trans: true => I:a; reset ; goto 1 true => O:b; reset Z ; goto 2 state: 2 invar: true trans: true => I:a; reset ; goto 2 true => Vuelta2; reset Z ; goto 0

Tabla A.1: Los tres componentes del ejemplo de la figura 4.2

Bibliografía

[ACD93] A. Alur, C. Courcoubetis, and D. Dill. Model checking in dense real-time systems. *Information and Computation*, 104(1):2–34, 1993.

[ACH+92] R. Alur, C. Courcoubetis, N. Halbwachs, D. Dill, and H. Wong-Toi. An implementation of three algorithms for timing verification based on automata emptiness. In *Proceedings of IEEE Real-Time Systems Symposium*, 1992.

[AD94] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[BCMD90] J. R. Burch, E. M. Clarke, K. L. McMillian, and D. L. Dill. Symbolic model checking. 10^{20} states and beyond. In *Proc. 5th IEEE Symp. Logic in Computer Science*, pages 428–439. IEEE, 1990.

[BF99] V. Braberman and M. Felder. Verification of real-time designs: Combining scheduling theory with automatic formal verification. In *Proc. 7th European Conf. on Software Eng./ 7th ACM SIGSOFT Symposium on the Foundations of Software Eng.*, LNCS 1687, pages 494–510. Springer Verlag, Sep 1999.

[BLL+96] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal- a tool suite for the automatic verification of real-time systems. In *Proceedings of Hybrid Systems III*, LNCS 1066, pages 232–243, 1996.

[Bra00] Victor Braberman. *Modeling and Cheking Real-Time System Designs.* PhD thesis, Facultad de Ciencias Exactas y Naturales, Universidad de Buenos Aires, Departamento de Computación, 2000.

[BS00] S. Bornot and J. Sifakis. An algebraic framework for urgency. To appear in Information and Computation, 2000.

[BW95] Burns and Wellings. *HRT-HOOD: A strutured Design Method for Hard Real-Time ADA Systems.* Elsevier, 1995.

[BW96] Burns and Wellings. *Real-Time Systems and Programming Languages.* Asisson Wesley, 1996.

[CE81] E. Clarke and E.A. Emerson. Design and syntesis of syncronization skeletons using branching-time temporal logic. In *Workshop on Logic Programs*, LNCS 131, 1981.

[CGL94] D. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *Proc.Principles of Programming Languages*, 1994.

[CW96] E. Clarke and J. Wing. Formal methods: State of the art and future directions. Technical Report CMU-CS-96-17, Carnegie Mellon University (CMU), Sep 1996.

[Daw98] C. Daws. Optikron: a tool suite for enhancing model-checking of real-time systems. In *Proc. of the 10th Conference on Computer-Aided Verification*, Vancouver, Canada, 28 June - 2 July 1998. Springer Verlag.

[dNMV90] R. deÑicola, U. Montanari, and F. Vaandrager. Back and forth bisimulations, 1990.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In *Proc. of Hybrid Systems III*, LNCS 1066, pages 208–219. Springer Verlag, 1996.

[DT98] C. Daws and S. Tripakis. Model–checking of real–time reachability properties using abstractions. In B. Steffen, editor, *Proc. of TACAS'98*, Lisbon, Portugal, 31 march - 2 April 1998. Springer Verlag, LNCS 1384.

[DY96] C. Daws and S. Yovine. Reducing the number of clock variables of timed automata. In *Proc. 1996 IEEE Real-Time Systems Symposium, RTSS'96*, Washington, DC, USA, december 1996. IEEE Computer Society Press.

[ECJ97] W. Elseaidy, R. Cleaveland, and J. Baugh Jr. Modeling and verifying active structural control systems. *Science of Computer Programming*, 29(1–2):99–122, Jul 1997.

[GSSAL94] R. Gawlick, R. Segala, J. Sogaard-Andersen, and N. Lynch. Liveness in timed and untimed systems, 1994.

[GW89] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics (extended abstract). In

G.X. Ritter, editor, *Information Processing 89*, pages 613–618, 1989.

[HHWT95] T. Henzinger, P. Ho, and H. Wong-Toi. A user guide to hytech. In *Proceedings of the 1st International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, LNCS 1019, pages 41–71. Springer verlag, 1995.

[HXSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *IEEE Information and Computation*, 111:193–244, 1994.

[Jos96] Mathai Jospeh. *Real Time System: Specification, Verification and Analisys.* International Series in Computer Science. Prentice Hall, 1996.

[KLK96] I. Kang, I. Lee, and Y.S. Kim. An efficient space generation for the analysis of real-time systems. In *Proceedings of the International Symposium on Software Testing and Analysis*, 1996. Also in *Trans. on Software Engineering* 1999.

[KTAB96] R. P. Kurshan, S. Tasiran, R. Alur, and R. K. Brayton. Verifying abstractions of timed systems. In *Proc. of the 7th Int'l. Conf. on Concurrency Theory*, LNCS 1119. Springer Verlag, 1996.

[LW97] Kim G. Larsen and Yi Wang. Time-abstracted bisimulation: Implicit specifications and decidability. *Information and Computation*, 134(2):75–101, 1 May 1997.

[Mil89] Robert Milner. *Communications and Concurrency.* Prentice Hall, 1st edition, 1989.

[NRSV90] X. Nicollin, J.L. Richier, J. Sifakis, and J. Voiron. ATP: an algebra for timed processes. In M. Broy and C.B. Jones, editors, *Proceedings of the IFIP TC 2 working conference on Programming Concepts and Methods*, Sea of Galilee, Israel, Apr 1990.

[TKY+98] S. Tasiran, S. P. Khatri, S. Yovine, R. K. Brayton, and A. Sangiovanni-Vincentelli. A timed automaton-based method for accurate computation of circuit delay in the presence of crosstalk. In *FMCAD'98*, 1998.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulation. In *Proc. CAV 96*, Jul 1996.

[TY99] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulation. *Formal Methods in System Design*, Feb 1999.

[Č92] K. Čerans. Decidability of bisimulation equivalence for parallel timer processes. In *CAV 92: Automatic Verification Methods for Finite-state Systems*, 1992.

[Yi90] Wang Yi. Real time behaviour of asynchronous agents. In *Proceedings of CONCUR90 (International Conference on Concurrency Theory)*, LNCS 458, Amsterdam, 1990. Springer Verlag.

[Yov93] Sergio Yovine. *Methodes et outils pour la verification symbolique de systemes temporises.* PhD thesis, Institut National Polytechnique de Grenoble, Grenoble, May 1993.

[Yov98] S. Yovine. Model checking time automata. In *Embedded System*, Lectures Note in Computer Sciences 1494. Springer Verlag, Oct 1998.